MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INVESTIGATIONS INTO THE PERFORMANCE OF
A DISTRIBUTED ROUTING
PROTOCOL FOR PACKET SWITCHING NETWORKS

by

Anthony W. Lengerich

December 1982

Thesis Advisor: J. M. Wozencraft

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A126 790	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Investigations into the Performance of a Distributed Routing Protocol for Packet Switching Networks		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; December 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Anthony W. Lengerich		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December 1982
		13. NUMBER OF PAGES 157
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Packet Switching Networks; Packet Radio; Routing Protocols; Dynamic Routing Protocols; Distributed Routing Protocols; Network Protocols; Protocol Simulation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Packet switching communication networks employ routing protocols to determine the path traversed by each packet as it passes through the network. Routing protocols which are adaptive and can restructure the packet paths in response to localized network congestion are called "dynamic" routing protocols. Dynamic routing protocols seek to optimize the routing (provide the shortest path) for each packet in the network. Routing protocols which are unaffected by the addition or deletion of any subset of

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

1

nodes are called "distributed" routing protocols. The relative
performance of two distributed, dynamic routing protocols is
determined using computer simulations. The protocols are
implemented of four theoretical networks and tested under a
range of network traffic loads.

Accession For
NTIS GRA&I	X
DTIC TAB	
Unannounced	
Justification	

By
Distribution/
Availability Codes
Avail and/or
Dist Special

A

Investigations into the Performance of a Distributed
Routing Protocol for Packet Switching Networks

by

Anthony W. Langerich
Lieutenant Commander, United States Navy
B.A., University of Colorado, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1982

Author _____

Approved by: _____
                                    Thesis Advisor

_____
                                    Second Reader

_____
Chairman, Department of Electrical Engineering

_____
                    Dean of Science and Engineering

3

ABSTRACT

Packet switching communication networks employ routing protocols to determine the path traversed by each packet as it passes through the network. Routing protocols which are adaptive and can restructure the packet paths in response to localized network congestion are called "dynamic" routing protocols. Dynamic routing protocols seek to optimize the routing (provide the shortest path) for each packet in the network. Routing protocols which are unaffected by the addition or deletion of any subset of nodes are called "distributed" routing protocols. The relative performance of two distributed, dynamic routing protocols is determined using computer simulations. The protocols are implemented on four theoretical networks and tested under a range of network traffic loads.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I wish to express my sincere thanks to Professor John M. Wozencraft for his guidance and patience during the course of this effort. I am indebted to him for both the concepts and the methods of research presented herein.

I dedicate this work to my wife, Linda, whose unselfish love and support made it all possible.

I. INTRODUCTION

A. THE PACKET SWITCHING CONCEPT

The purpose of any communications network is to provide paths for the transmission of information. It is desirable that the process which controls the transmission and reception of messages within the network be transparent to the user. To achieve this result it is required that the communication network impose a tolerable delay in the relay of messages between the originator and the destination. Further, the network should require a minimum number of instructions from the originator to accomplish the task. Ideally, the originator would be required to provide only the destination of the message to the network. The communications network should then be capable of transmitting messages to their destinations in as short a time as possible.

For communications networks which serve many users and have the requirement to exchange large amounts of data, the concept of "packet switching" may provide a method of operation which will satisfy both the speed of service and the transparency requirements of the user. In addition, an increased robustness of the network can be realized without

11

creating redundant cicuitry if an appropriate routing algorithm is employed within the network.

The packet switching concept may be broadly described as the coordinated movement of specified length data streams within a communications network such that the data streams always travel the least congested (ie. quickest) route between the originator and the destination. Packet switching envisions the breakdown of a message into fixed length segments (packets) which are transmitted through the network to the destination by the "shortest path" (called the best path) which is known to exist at the time the packet is released into the network. The implementation of the packet switching concept requires that each station (node) in the network be capable of processing certain data concerning the structure of the network. This is necessary so that the correct routing can be assigned to each packet as it is released by the node into the network. This process can be implemented in the computer software which would control a node's participation in the network. It is important to recognize that the routing of different messages is not independent of each other, since (for example) sending all traffic via one particular link will

introduce congestion (queueing) delays that might be avoided by distributing the traffic over a variety of available links.

Packet switching techniques can be used to transmit both data and digitized voice signals. While the transparency of service to the data user is resolved through use of high bit rate transmission systems and storage buffers at each node, the real time requirement for intelligible voice communications may require the establishment of a dedicated circuit for each conversation. It may also be possible to establish a dedicated voice transmission circuit each time the transmitter is "keyed" rather than maintain a circuit for the duration of the conversation, or to always transmit over the same route (called a "virtual circuit") and to share the channel capacity with other messages during interstices in the conversation. In this way the amount of time a particular set of links remains dedicated to a single transmission is minimized.

B. NETWORK PROTOCOLS

Central to the operation of any communications network is the set of instructions which directs the exchange of network information between nodes. These instructions may

13

be implemented in hardware or software and are referred to as Network Protocols. The information contained in messages generated by these protocols is used by the communications network to implement the routing of user message traffic within the network. Network Protocols are commonly divided into several hierarchical levels according to their particular function within the network. For instance, the International Standards Organization adopted a "Reference Model of Open System Interconnection" (ISO) which consists of seven layers of protocol. The different layers as described by Tanenbaum [Ref. 1], are: physical layer, data link layer, network layer, transport layer, session layer, presentation layer and application layer. Heritsch ,in Reference [2], addresses three levels of protocol which are of concern in studying Distributed Routing Protocols. Five of these layers are incorporated into the three protocol levels discussed by Heritsch. The interrelation of these levels and layers is presented in the following paragraphs. Two layers, the session layer and the application layer, are not considered since they are not present in the the test networks used in this study.

The lowest level of protocol is called the Node-to-Node protocol and consists of those instructions necessary to ascertain and verify that a usable communications path exists between any two adjacent nodes. This level is a combination of the physical layer and the data link layer of the ISO and serves to establish the physical exchange of data between nodes. In addition the Node to Node protocol specifies the "handshake" and "acknowledgement" processes which are required whenever messages transit between two neighboring nodes. (The term "neighboring nodes" implies that a direct two way communications path exists between adjacent nodes. Such a path is referred to as a "link".) If the computer software within the node is configured to monitor the transmission and receipt of Node to Node protocol messages, then the continuing function of verifying that a usable link exists can be accomplished without another form of protocol message being entered into the network.

The next level of protocol is the Network Management Protocol which corresponds directly to the network layer of the ISO and exchanges all the information necessary for the nodes to make decisions as to which route each message

originated by or relayed by the node should travel enroute to its destination. There are several different concepts of Network Management protocol. Since if the amount of Network Management traffic is large there is less residual capacity in the network for the transmission of user generated traffic, it is essential to establish an understanding of the portion of total network capacity required to implement a given Network Management Protocol. This study is concerned with developing such an understanding for the Network Management Protocol proposed by Heritsch [Ref. 2].

The final level of protocol is the User Service Protocol. This level is equivalent to the presentation layer of the ISO. As the name implies, this level is concerned with the interface between the user and the communications network. Functions accomplished by this protocol level may include the automatic breakdown of user messages into "packets", attaching accounting numbers to the messages and releasing or receiving messages into and from the network. At this level, tradeoffs made between network transparency and the required user interaction with the network are most apparent.

C. DISTRIBUTED ROUTING PROTOCOLS

The singular feature which distinguishes a packet switching communications network from other communications networks is the repackaging of user messages into smaller fixed length "packets", each of which can be treated separately by the network. An advantage of using fixed length packets is that the network can seek to transmit packets via routes which currently have the fewest number of packets flowing through them. Since all packets are the same length a simple count of the number of packets waiting in queue for a specific link is a measure of the delivery delay a packet traveling that route would experience. The packet switching network attempts to minimize the delay time by routing packets over links which have the fewest number of packets active within them. It is desirable to use Distributed Routing Protocols in packet switching networks to decide which routes are shortest.

Distributed Routing Protocols are sets of instructions utilized by the network to determine on the basis of information exchanged with its neighbors which path each packet should take from its originating node to its destination node. Distributed Routing Protocols may be either synchronous or asynchronous in design.

Synchronous Distributed Routing Protocols cause all nodes within the network to change their routing instructions simultaneously and at periodic intervals. Asynchronous Distributed Routing Protocols permit the nodes to adopt new routing instructions independently whenever the node finds a path to the destination node which is shorter than the path it had previously been using.

This study examines the Distributed Routing Protocol proposed by Heritsch, [Ref. 2], determining its performance in both the synchronous and asynchronous forms and comparing its performance to the centralized routing protocol based upon the Djikstra shortest distance algorithm [Ref. 3]. The protocols are examined using a computer simulation which uses several different networks and several levels of network message loading to provide an indication of protocol performance under a variety of conditions.

II.   A NAVAL APPLICATION

The concept of packet switching offers several
advantages to users who require high data rate transmission
capability and a robust communications network through which
large volumes of message traffic must pass.   In particular
the requirements of the Navy Tactical Data System (NTDS)
might in the future be met utilizing a packet switching
network.

NTDS is a computer based tactical information display
and decision system . NTDS information is exchanged between
units of the fleet via a radio frequency digital
communications network.   (The term "fleet" is used here in
its broadest sense and may include such diverse units as
ships , all types of aircraft and troops ashore.)   In its
present form, the NTDS utilizes a centralized communications
network with a net control station (master node)
coordinating the exchange of information between units of
the fleet.   Participating fleet units (nodes)   which have
direct links with the originating node (ie. are within radio
reception range)   receive the transmitted message.   Nodes
which are beyond the radio transmission range must rely on
retransmission of the information by a node which is

directly linked to a sending node. Nodes which retransmit messages may be thought of as sub level master nodes and their loss may deprive outlying nodes of information and direction.

In contrast, the packet switching concept offers a distributed communications network to support the information and coordination requirements of a widely dispersed fleet. By employing a Distributed Routing Protocol, a packet switching network can achieve increased network robustness by decentralizing the control of the network and seeking to optimize the routing of each message entered into the network. An NTDS participant would enter information into the communication network addressed to other NTDS participants as required. Upon entry the information would be reformatted into "packets" and transmitted along the best path to the destination nodes. Since all nodes are constantly "updating" their status and the status of their connecting links to other nodes in the network, the best path between any two nodes can be autonomously and continuously determined by each individual node using the network's Distributed Routing Protocol. The loss of nodes or degradation of links within the network

becomes known to all nodes and best paths can be reevaluated such that the network can continue to pass messages as long as possible. This feature of a packet switching network derives not from the physical addition of links between nodes but from considering of all nodes and links within the network as potential transmission paths to any other node in the network.

An important result of the packet switching concept is its ability to automatically relay messages to nodes which are beyond the radio reception range of the sending node. Since the sending node provides the destination addresses at the time of message transmission, the network can assume responsibility for delivery of the message rather than the conventional method of broadcasting a message and maintaining the receiving nodes within direct radio reception range. This capability would allow widely dispersed nodes to enjoy the information and coordination advantages of the NTDS without the burden of additional radio circuits. Additionally, the packet switching concept is adaptable to a variety of Anti-Jam (AJ) and Low Probability of Intercept (LPI) communications schemes which are under study or development by the Department of Defense.

In particular, packet switching is ideally suited for use with several spread spectrum communication schemes.

Conceptually, the most difficult problem to be overcome in the deployment of packet switching networks afloat is the mutual interference which two nodes transmitting simultaneously in the same frequency band might cause at the receiving node. This difficulty could be overcome using a "carrier sense" or a "listen before transmit" approach in the transmitter/receiver design and employing a signalling scheme which permits the receiver to "lock up" on the first signal received to the exclusion of all other signals. Conceivably, 'units could operate at slightly different carrier frequencies and the receivers would automatically shift their intermediate frequency sections to "lock up" on the sending unit's carrier. Mutual interference effects can also be ameliorated by use of wide-band psuedo-noise transmissions using uncorrelated keystreams.

It is with the possibility in mind, that the packet switching concept could improve the usability and survivability of NTDS, that this study was undertaken. To this end a set of generalized network configurations were developed so that several performance characteristics of the

packet switching concept could be determined. The specifics
of the networks are outlined in Chapter III.

III. PARAMETERS OF A PACKET SWITCHING NETWORK

A. QUANTIFYING PROTOCOL PERFORMANCE

Critical to determining the relative performance of different routing protocols operating within a packet switching network, is the determination of an expression (mathematical or otherwise) which provides an unbiased and meaningful basis for comparison. In this study the average delay time in delivering a packet (T_p) is used as one measurement of protocol performance. Another measure of performance used is the utilization factor of the network (p). A mathematical derivation of the inter-relation of these terms, based upon queueing theory, is presented in Appendix A.

The average time delay per packet, (T_p) has the advantage of relating network congestion to network loading in terms of delay time, which is ultimately the most meaningful statistic to the network users. The utilization factor provides a measure of the fraction of the capacity of the network which is required to transmit user messages (packets) or network management messages (updates) given a specified network and traffic loading for the network. Both of these measures are affected by several physical

24

parameters of the network. For instance the processing time within a node for a packet adds delay time to the packet which is proportional only to the number of nodes traversed by the message route, hence independent of the routing protocol in use. On the other hand, if a packet must wait in a queue for an update to be transmitted on a link then the incurred delay is of interest because it represents interference by network management traffic with the transmission of user message traffic. A discussion of the various network parameters and their impact on the comparison measures, average delay per packet and update utilization factor, is presented in the next section.

B. NETWORK CONFIGURATIONS

In order to ensure that the performance measures "T_p" and "p" obtained from the simulation of the network are accurate, it is necessary to determine which physical characteristics of the network effect the measures of protocol performance. By fixing these characteristics as constants for all the networks under consideration, it is possible to obtain measures of performance which are representative of the protocol's performance in networks of varying complexity and traffic loading conditions. The

25

network characteristics which were deemed to be significant and the values assigned to them are listed in Table I and are discussed in the paragraphs that follow.

TABLE I

Significant Network Characteristics

Parameter	Value
packet transmission time for any link	0.05 sec.
packet processing time in any node	0.0001 sec.
update transmission time for any node	0.02 sec.
update processing time in any node	0.00001 sec.
update cycle time	0.2 sec.
update channel value calculation window	0.5 sec.
number of nodes in the network	variable
number of links in the network	variable
average number of packets per user message	variable
average number of user messages per sec.	variable
simulation run time	30 sec.

The packet transmission time is related to the length of the packet and the data transmission capacity of the link over which it is to travel.   For this study only two types of messages (updates and user messages) are allowed to enter the links.   User messages are subdivided into packets which are fixed in size to be 2.5 times longer than an update

message. In this way the length of an update is made to be small compared to the length of the packet. This action appears reasonable in view of the significantly smaller information content in an update versus a packet. The relative length of the the updates is also important to the performance of the distributed routing algorithm when computing channel values which are discussed in section C of this chapter. The time of 0.05 seconds selected for the transmission time of a packet, is equivalent to a packet length of 800 bits transmitted over a link with a capacity of 16,000 bits per second. Alternatively, it equates to a 120 bit packet transmitted over a link with a capacity of 2,400 bits per second.

The time required to process a packet within a node was arbitrarly chosen to be 0.0001 second and the processing time for an update was set to be one tenth of the packet processing time. This assignment also appears reasonable since in processing a packet each node must read the complete address section of each packet to determine onward routing if required and prepare a new address section for each packet which is retransmitted. On the other hand, an update message contains only a relatively small amount of

information which is either discarded or stored and relayed as required.

The update cycle time represents the frequency with which the network establishes a new best path for routing packets. An asynchronously operating protocol does not implement best paths simultaneously throughout the network but rather each node implements a new best path immediately upon determining that a new best path exists. However, if a distributed routing algorithm is implemented synchronously, it is necessary for the update message to have reached all the destinations before a new routing scheme is established in the network. An update cycle consists of two equal length time periods. During the first period nodes generate the required update messages, which receive transmission priority over packets. During the second period nodes receive and relay update messages as required. Nodes are restricted from generating update messages during the second half of the update cycle in order to allow the updates to suffuse throughout the network. If the nodes were an average of 16.4 kilometers (10 miles) apart and a maximum of 5 relays were required for each update to reach the destination node most distant from it, then the maximum time

required for an update to reach its destination can be
computed as

        time required = travel time + processing time
                      + transmission time

which becomes

$$t = 5 * \frac{16.4 \text{ km}}{3 * 10^{**}5 \text{ km} / \text{sec}} + 5 * 0.00001 + 5 * 0.002$$

or

        t = .025 sec.

which is well within the available 0.1 seconds. Note that
this result implies that for applications where the nodes
are close (in relation to signal travel time) then the
propogation time is insignificant compared to the time
required to process an update through a node. By choosing
an update cycle time of 0.2 second the network
configurations used in the simulation could be considered
representative of real world networks which have longer link
lengths and/or a greater number of relays required for the
delivery of an update message.

        It is the complexity of the network which has the most
potential for impacting on the performance of a Distributed

Routing Protocol, and it is therefore important to consider how the topology of the network interacts with the protocol. The number of nodes in a network determines the number of update messages which must be generated during each update cycle, which translates into an increasing requirement for network capacity to pass updates as the number of nodes increases. This results in less network capacity available for the transmission of user messages. The number of links in a network is also important since it determines the capacity of the network both for message and update traffic.

Traffic loading is another network parameter which may affect the performance of a Distributed Routing Protocol. For a packet switching network the traffic loading is controlled by three factors, the number of user messages entering the network per unit time and the length of the messages and the distribution of the originator and destination addresses of each message. Since messages are decomposed into fixed length packets, this last factor is the same as the number of packets in a user message.

In a real world network each of these parameters is random and it is common in queueing theory to assign a distribution to the frequency with which messages are

entered into the network and the number of packets into
which a message is decomposed in terms of a probability
distribution.   It is often assumed that generation of
messages into a network can be represented by an exponential
distribution and that the number of packets in a message can
be represented as a uniform distribution.   Two mean values
of packet distribution were used in this study (5.5 and 10.5
packets per message)   so that the effects, if any, of the
length of the messages on the performance of the protocols
could be determined.

A final factor which may affect the performance of a
Distributed Routing Protocol is the simulation run time. If
the complete network is considered to be a system and the
first message into the system is viewed as an impulse input
into the system then it is clear that the system will
undergo some period of transition between the dormant
condition (ie.   no inputs)   and its equilibrium operating
state.   The length of this transition phase can be
determined mathematically for the probabalistic case of a
communications network only with difficulty.   However, by
observing the results of the network simulation after
several different simulation run times it was determined

that a run time of 30 seconds allowed the transition phase to be completed and provided a period of steady state operation long enough so that meaningful data could be taken.

A total of four networks were generated for use in this study. A diagram of each network is provided in Appendix B. The number of nodes in each network varied between 5 and 20 and the number of links varied between 10 and 40. To reduce distortion of the simulation results which could be caused by different network topologies, a base network of 5 nodes and 10 links was adopted. By expanding the base network in a consistent manner, valid comparisons between the performance of a protocol and network complexity could be drawn. More complex networks were created by expanding the link/node structure of the base network. The chosen base network is illustrated in Figure 3.1 and consists of five nodes and ten links.

To provide a convenient method of referencing the networks the notation "5/10" has been adopted to denote a 5 node and 10 link network. The 5/10 network was expanded into a 10/20 network by adding an additional 5 nodes and 10 links. Similarly, a 15/30 and 20/40 network were derived.
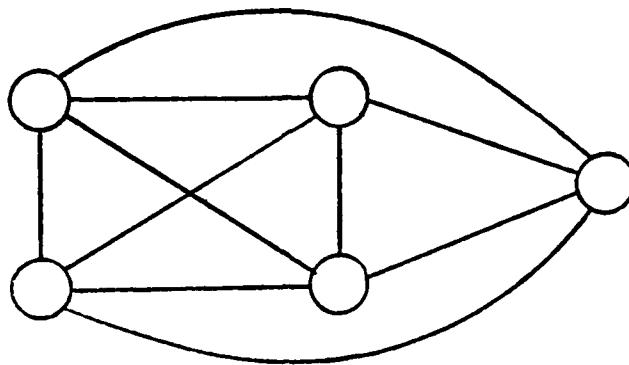
Fig. 3.1.   The 5/10 Network.

Note that each succeeding network contains the preceeding network.

C.   CHANNEL VALUES

As described in section B of this chapter, the channel values ("link distances" in some literature) are computed during the update window time and represent the current demand for a particular link's services as a relay path. The channel value also represents a prediction of the demand on a particular link during the upcoming update cycle. By using these channel values it is possible to determine a path between any two nodes which will have the lowest cumulative channel value (ie. the best path during the next update cycle).   It is the function of the distributed

33

routing protocol to generate, transmit and evaluate the channel values and establish best paths within the network. To accomplish this task the protocol generates update messages which contain channel value information. The exact content, method of relay and method of evaluating the update message is unique to each protocol.

The method of generating channel values for both the Heritsch and Djikstra algorithms as implemented in this study is identical, and a complete description can be found in Heritsch [Ref. 2]. In short, the channel value represents the average number of packets waiting in the queue to be transmitted over the link during the preceding 0.5 second of simulation time.

IV. TWO DISTRIBUTED ROUTING PROTOCOLS

A. THE HERITSCH ALGORITHM

Heritsch proposed in Reference [2], a Distributed Routing Protocol which seeks to minimize the number of update messages required to be circulated througout a network during a single update cycle. Update messages in the Heritsch algorithm contain three items of information: the node which just relayed the update, the node which originated the update and the cummulative channel value from the node which just received the update to the node which originated the update, through the node which relayed the update. Thus each node receiving an update can compare the cummulative channel value through the relaying node to the cummulative channel value of the path over which it is currently passing packets to the node which originated the update. If the new channel value in the most recent update is less, a new best path to the node which originated the update is adopted through the node which relayed the update. In any event, the node which just received the update now generates an update message to all of its neighbor nodes (less the node which relayed the update that started it all), and informs all the neighboring nodes of the new

35

cummulative channel value. These nodes in turn repeat the process of evaluting the information contained in the update, establishing new best paths if the minimum cummulative channel value changes.

If no new best path is adopted and the channel value through the current best path has not changed, the node discards the update without relaying the information it contained. Since nodes which are "upstream" from the node which received the update do not change their best paths unless the best path "downstream" changes, there is no reason to relay the update further through the network. Note that even if a new best path is not adopted but the channel value changes over the current best path, this information must be relayed "upstream" since it may affect the best path decisions of "upstream" nodes. What is significant is that the algorithm purges update messages from the network at the first opportunity and thus releases channel capacity to the network for the relay of packets.

The disadvantage of the algorithm is that it does not guarantee optimum routing for each packet released into the network. It is possible for a packet to loop through several nodes before being delivered to its final

destination. This looping is the result of a packet following a best path which is changed more than once during the transit of the packet through the network.

The basic algorithm as stated above is substantially identical to the routing procedure originally used in the ARPANET. The major innovation proposed by Heritsch is that the number of update messages required in a given network could be reduced if the network was subdivided into nodal colonies which are called groups and families. A group can be defined as a collection of nodes and a family is a collection of groups. By treating the groups and families of nodes as "super nodes" it may be possible to significantly reduce the number of update messages required to implement the protocol. This reduction is accomplished by restricting the propagation of the update message to the basic group of the node which originates the update. Nodes which are connected to other groups or families originate updates which reflect the cumulative channel values from that node into the neighboring family or group. The result is that all nodes within a basic group can determine the best path to any other group or family and are not required to determine best paths to all other nodes in the network.

37

Thus a packet generated in one group would travel the best path to the destination group and once it reachs the destination group receive onward routing to the destination node.

The Heritsch algorithm may be implemented either synchronously or asynchronously in a packet switching network. Since the algorithm allows each node to autonomously determine new best paths it is possible to implement new best paths whenever an update is received rather than adopting best paths in synchronism with all the other nodes in the network. The only constraints to the algorithm are that update message generation be restrained to a fixed time interval so that update messages are able to reach their "final" destinations before a new update generation cycle starts.

A comparison of the algorithm's performance when implemented both synchronously and asynchronously in identical networks was accomplished as part of this study. No significant difference was found to exist in the performance of the protocol under the two implementations.

From the viewpoint of network robustness, it is desirable to implement the algorithm asynchronously.

However, for purposes of comparison with the Djikstra algorithm, (which can only be implemented synchronously), the Heritsch algorithm was restricted to the its synchronous implementation.

B. THE DJIKSTRA ALGORITHM

The algorithm proposed by E. Djikstra in Reference [3] is used in the current ARPANET routing protocol, which requires that each node within the network receive the channel value for each link in the network before a new best path is determined. The protocol causes each node to generate an update which contains the channel value from the originating node to each of the originating node's neighbors. The data is forwarded to each node in the network and each node enters the received channel values into a matrix which is manipulated in such a way that a best path is produced from each node to each other node in the network. The Djikstra algorithm may also be adapted to utilize the group and family scheme proposed by Heritsch. An examination of the Djikstra algorithm's performance using a group and/or family configured network was not conducted as a part of this study.

The significant differences between the two protocols are the number of update messages required to implement each protocol and the "quality" of the best path which is determined by each algorithm. The Djikstra algorithm provides an optimum quasi-static best path for each packet released into the network. That is to say, each node in turn routes the message to the next relay node along the best path in accordance with the Djikstra algorithm which is guaranteed to produce a minimum distance route. Similarly, in the Heritsch algorithm each packet is addressed to the next node along its estimated best path, but there is no guarantee that this esitmated path is indeed the optimum path. With regard to the number of update messages required by each algorithm, at first glance it would appear that the Djikstra network requires an inordinate amount of network capacity to pass updates through the network. The situation is not this bad however, since update messages which would travel along the same best path to their destination can be combined to reduce the total number of update messages required.

The matrix manipulation required by the Djikstra algorithm to determine the best paths is accomplished in the

following manner. If the 5/7 network illustrated in Figure
4.1 is considered as an example then the channel values
between adjoining nodes may be represented by the symbols 7>
or <5, where the arrowhead indicates the direction of the
channel value and the number indicates some relative measure
of the channel values as they might appear at the start of
any update cycle.
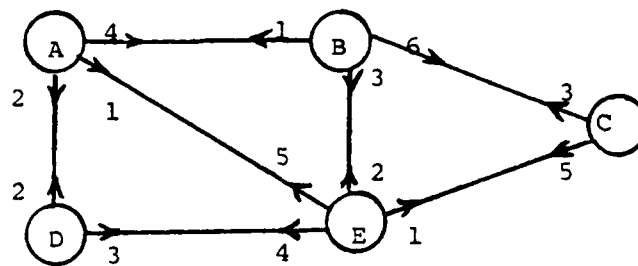
Fig. 4.1. A 5/7 Network

From this network the matrix in Figure 4.2 is constructed.
Note that nodes which have no direct link between them are
initially represented at the start of every cycle as having
an infinite channel value between them.

Beginning with node A, observe that a direct link exists
with node B and that the channel value from node A to node B

Fm	A	B	C	D	E
A	–	4		2	1
B	1	–	6	∞	3
C	∞	3	–	∞	5
D	2	∞	∞	–	3
E	5	2	1	4	–

**Figure 4.2.   Initial Channel Value Matrix For The Djikstra
Algorithm.**

is 4 (the notation A/B/4 will be used as a convenient
description of the path and its channel value).   Next every
other path in the network from node A to node B is
considered in turn to determine if a cummulative channel
value less than 4 can be found.   First the A/C/∞ + C/B/3
path is considered which represents a path from node A
through node C to node B.   Clearly the result of this path,
A/C/B/∞ , is greater than the A/B/4 path and so a new best
path through node C is rejected.   Next the A/D/2 + D/B/∞ =
A/D/B/∞ path is considered and also rejected as a new best
path.   Finally, the path A/E/1 + E/B/2 = A/E/B/3 is
considered and adopted as the tentative new best path from
node A to node B since the resultant cummulative channel
value of 3 is less than the previous value of 4.

Similarly the channel values from node B to each other node are considered and tentative best paths established. When all paths between nodes of the network are evaluated and tentative best paths established (ie. the matrix is run through completely one time) the entire process is repeated until a run through the matrix is completed with no new tentative best path adopted. When this occurs the best paths within the matrix are adopted simultaneously by all nodes of the network for the duration of the next update cycle and represent the optimum paths between nodes.

	To				
Fm	A	B	C	D	E
A	–	3	2	2	1
B	1	–	3	3	2
C	4	3	–	6	5
D	2	5	4	–	3
E	3	2	1	4	–

Fig. 4.3. The Completed Best Path Matrix

The final result of the matrix manipulation for the update cycle considered in this example is shown in Figure 4.3.

V. SIMULATION OF THE PROTOCOLS

A. PROGRAMMING SCHEME

The simulation programs used in the study were written in the SIMSCRIPT II.5 programming language [Ref. 4]. This language provides an excellent vehicle for simulation work in which many events may occur almost simultaneously. In addition, the ability to control the flow of the simulation by utilizing a language generated simulation clock, provides the user with a means of obtaining data from the simulation at various times without regard for the events being processed.

Three simulation programs were developed from the program presented by Heritsch in Reference [2]. The three programs are listed at the end of this study and represent an asynchronous implementation of the Heritsch algorithm, a synchronous implementation of the Heritsch algorithm and a synchronous implementation of the Djikstra algorithm. The program lisitng for the synchronous implementation of the Heritsch algorithm is provided in Appendix C. Each simulation is constructed in modules which correspond to the different functions which the network and the routing protocol must perform. For example, each simulation

44

contains individual subroutines for the generation of updates, arrival of packets within a node and the computation of channel values. A listing of all the arrays used within the programs is provided in Appendix D to assist in tracing the functional composition of the programs. The parameters of the network under consideration are externally input into the simulation as a data set which is read by the source code during the execution of the simulation. A sample of an input data set is listed in Appendix E.

The results of the simulations can be output in several formats. The most extensive output available is a complete listing of every event occuring during the simulation. It is possible to limit the output to only the desired data by specifying the desired results in the "Special Output" subroutine of each program. The simulation programs also feature the ability to make any number of simulation runs with various input parameters during a single job submission. In addition, the results of the simulation can be diverted to either mass storage peripheral devices or the normal printer output or both. A sample of the output format used for this study is provided in Appendix E.

The implementation of the packet switching network is
identical in all the simulation programs. Each link within
the network is "configured" to have two input queues (a
packet queue and an update queue). No queues are
constructed for the input side of each node. Nodes are
required to give priority to the transmission of updates
over packets as would be required in a real world network.
Update messages are generated only during the first half of
the update cycle. The precise time at which each node
generates its update message is uniformly distributed
throughout the first half of the update cycle. User
messages are generated into the network at time intervals
which are exponentially distributed. The originating node
and destination node of the message and the number of
packets in the message are determined using uniformly
distributed random variables. The destination node of a
user message can be constrained to be outside the
originating node's group and/or family in order to ensure
that a specified percentage of the user messages are
destined for other groups and/or families.

B. ARTIFICIALITIES OF THE SIMULATION PROGRAM

Several artificialities were introduced into the simulation programs in order to reduce computer run time and program complexity. The most significant of these is the assumption that each node has a processor dedicated to each link entering the node. This implies that the node can simultaneously receive and process inputs on all of its links. While the capability of parallel processing is desirable in a packet switching network, it is not a requirement and may be replaced by employing a "listen-before-transmit" and "receiver-lock-up" node design, in which each node can communicate with only one other node at a time. In either case the maximum throughput of any node is considered to be determined by the output capacity of the node and not its input capacity. Thus the absence of input queues in the simulation is considered reasonable.

The simulation also does not consider the network capacity which is required to implement the Node to Node Protocol, which is vital to the operation of a packet switching network. The capacity required to carry the Node to Node protocol messages is relatively insignificant compared to that required for update and packet

47

transmissions. Since this study is concerned only with the performance of the routing protocol, the network capacity required by the Node to Node Protocol is disregarded.

C. LIMITATIONS OF THE SIMULATION PROGRAM

The detailed results of the simulations depend substantially on the "seed" numbers which are used to initialize the random number streams generated internal to the SIMSCRIPT language. Since the random number streams are used to determine the origination node, destination node, time of message origination and the number of packets in a message, the simulation will provide somewhat different results for the same input parameters if different "seed" numbers are used. The variability of a particular result from the average result computed from several different runs of the simulation using the same inputs but different "seeds", was empirically determined to be on the order of 5 percent of the average result. Thus, although the graphical results of the simulations (Appendix F) appear to be discontinuous in nature as the input parameters are changed, it is expected that these perturbations are due to the "randomness" of the simulation. If a sufficient number of simulation runs were made for each data point, the average

48

of these points would be expected to result in a "smoother" curve.

D. GENERATION OF DATA

A total of twelve different simulation runs were required to produce the results presented in this study. The physical parameters of the networks and the network loadings were selected to represent the most critical and interesting aspects of the interaction between the network and the routing protocol in use. The Heritsch algorithm was synchronously implemented on four networks (5/10, 10/20, 15/30 and 20/40) and asynchronously on two networks (10/20 and 15/30). The algorithm was also synchronously implemented on the 10/20 and 15/30 networks, with the networks divided into 2 groups of 5 nodes and 3 groups of 5 nodes respectively, to investigate the capacity required to implement the algorithm in a multiple group/family network. Lastly, the Djikstra algorithm was implemented on the 5/10, 10/20, 15/30 and 20/40 networks.

The graphs of the data contained in Appendix H were derived from the data obtained from these twelve simulation runs and are presented in two forms. The first group of twelve graphs illustrates for each network and each

49

algorithm, the average delivery delay per packet versus the loading of the network. The second group of twelve graphs shows the packet and update utilization factors versus the network loading for each algorithm and each network. The interpretation of the data presented in the graphs is provided in Chapter VI.

VI. RESULTS AND CONCLUSIONS

A. RESULTS

1. Comparison of Average Delivery Delay

The average delivery delay for a packet (T_ρ), was computed using Equation (A-12) from Appendix A. This equation uses the measured utilization factors for updates and packets (p_u and p_ρ) and a specified network loading (g) to compute T_ρ. Although the equation does not provide an absolute measure of the protocol's performance it is useful in comparing the relative performance of the Djikstra algorithm to the Heritsch algorithm. Theoretically, T_ρ should increase dramatically as the network loading begins to approach the network capacity. Figure 6.1 illustrates the general form of the expected T_ρ versus g results. This expected result reflects the obvious conclusion that the delivery delay experienced by a packet must increase without bound when more packets are entering the network per second, on the average, than can be delivered by the network in a second.

From the simulation results contained in Appendix F, it is concluded that a packet switching network employing the Djikstra algorithm has less average delay per packet

than an identical network under identical loading which employs the Heritsch algorithm. Further, a network which employs the Djikstra algorithm is capable of accepting a wider range of network loading conditions before the average delivery delay per packet increases dramatically.
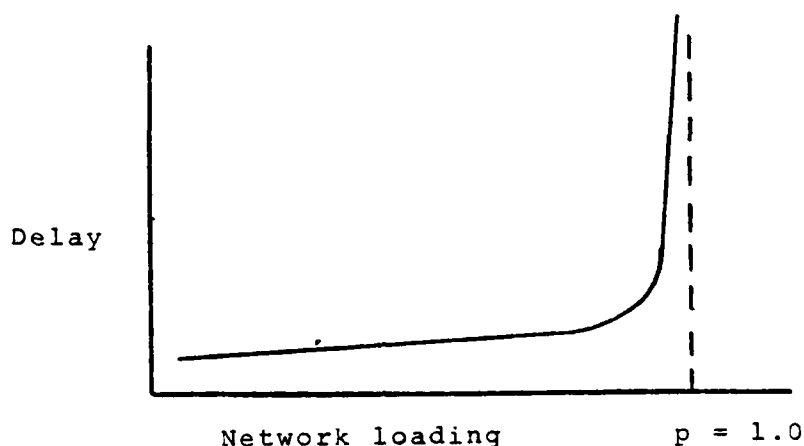
Fig. 6.1. Expected Average Delay Per Packet

The simulation results contained in the first twelve graphs of Appendix F present two measures of the average delay per packet. One curve on each graph represents the results of Equation (A-12) and the other represents the average actual delay per packet which was computed using measured data from the simulation as follows.

$$M_p = \frac{t_t}{k} \qquad\qquad (6\text{-}1)$$

where

M_p = the average delay per packet (measured)

t_t = the sum time required to transit the network of all packets which reach their final destination

k = the total number of packets originated during the simulation which reach their final destination.

Equation (6-1) provides a much different result for the average delay per packet than does Equation (A-12), and both of these methods of predicting the protocol performance are subject to considerable inaccuracy as discussed in the following paragraphs.

As noted in Appendix A, Equation (A-12) was derived from queueing theory results which apply to networks with a fixed routing and an exponential service time distribution. Since the networks under study have a dynamic routing scheme and a fixed service time, the direct transfer of mathematical analysis from one application to the other has questionable validity. Adding to this uncertainty is the priority handling of update messages and their shorter service time, which was factored into the derivation

53

presented in Appendix A. Finally, some error in the results
of the equation is expected due to the method of measuring
the utilization factors. Since it was desirable to avoid
including the transient effects of initial network start-up,
the measurement of the utilization factor for each link was
limited to the last half of the simulation. To accomplish
this restriction, it was necessary to structure the
simulation program so that any link which was busy at the
instant of "half time" was identified and the fixed value of
0.05 seconds (packet transmission time) was added to the
busy time of the link for the first half of the simulation.
This busy time was then subtracted from the total time the
link was busy throughout the simulation. The end result was
a measured utilization factor for any one link which could
be on the order of 0.003 percent smaller than the actual
value. Despite these probable inaccuracies, the results of
Equation (A-12) agree with the intuitive notion that the
average delay per packet should exhibit a dramatic increase
as the network loading approaches the network capacity.

In a network with fixed routing the "break point"
for the rise in delay time can occur at moderate network
loads if a critical subset of links is saturated ($p = 1.0$)

and a portion of the nodes are effectively "cut-off" from the network. The rational for employing a dynamic routing scheme is that saturated links are bypassed (wherever possible) until their utilization factors return to a more normal level. The effect is to postpone the dramatic rise in network delay until greater network loading conditions are experienced by the network. Thus a whole cutset of the network (vice only a few links) must have a utilization factor nearly equal to unity before the characteristic delay curve exhibits its predicted rise.

The actual measurement of the average delay per packet was accomplished by accumulating the transit time of all messages which reached their destination and dividing by the number of messages which reached their destination. This measure does not include a method of determining the delay near the "break point", since near that point the number of messages which are delivered in a finite time period becomes relatively constant regardless of the network loading. Thus the result from this method of computing delay does not exhibit the strong non-linearity of Equation (A-12) in regions near utilization factors equal to unity. Most packet are not delivered under these circumstances, but

accumulate in the queues instead.   If a correct expression
of the average delay per packet is to be measured, it must
account for the future delay which queued messages will
incur as they ultimately work their way out of the network.
The formulation of an expression which accurately describes
the shape of the delay curve in its entirety (including the
"break point") proved an elusive goal.   However, it can be
intuitively argued that the performance of the protocol in
regions near $p = 1.0$ must be such that the average delay per
packet is increasing very dramatically as the network
loading approaches close to the network capacity.   This
intuitive argument was verified by running the simulation
for a duration of greater than 180 seconds at network
loadings which produced average network utilization factors
in excess of 0.996 percent.   Under these conditions it was
found that the average delay per packet did exhibit the
expected dramatic increase.   (Note that this delay was
evident only after relatively long simulation run times.)

In summary, the theoretical delay curve predicted by
Equation (A-12) does not accurately predict the performance
of the Distributed Routing Protocol due to the adaptive
nature of the protocol and the finite duration of the

56

measurement period. Further, a successful mathematical model for predicting the measured performance of the protocols was not found.

Lastly, the average delivery delay which resulting when the Heritsch algorithm was implemented in a test network which had the nodes divided into groups was investigated. Although the average delay per packet was significantly decreased using the group/family routing feature of the Heritsch algorithm, for the small networks studied here it was not as low as the average delay per packet experienced in a single group network of the same size which employed the Djikstra algorithm. It is believed that more significant and favorable results will be achieved if larger networks are used in comparing the performance of the Heritsch algorithm in single group versus multiple group/family networks.

2. Updating Requirements Versus Network Complexity

The inherent disadvantage in using Distributed Routing Protocols is the requirement to utilize a portion of the network's capacity to support the updating requirements of the protocol. Naturally, the protocol which provides the most optimum routing and requires the least network capacity

to operate is the most desirable. The amount of network capacity required by the protocol is determined by the frequency at which update messages are generated and the number of nodes by which each update must be received.

It was determined, during the course of this study that the network capacity required by each of the protocols under consideration is a nearly constant value for a given network and did not depend, to any great extent, on the traffic loading of the network. Table II contains the average update utilization factor for the Heritsch algorithm (for single and multiple group networks) and the Djikstra algorithm (for single group networks only). No data was obtained for the Heritsch network implemented on a 5/10 or 20/40 multiple group network.

TABLE II

Update Utilization Factors

Network	Heritsch (single group)	Heritsch (multiple groups)	Djikstra (single group)
5/10	.044	****	.009
10/20	.085	.028 (2)	.019
15/30	.139	.041 (3)	.023
20/40	.201	****	.028

The numbers in parenthesis in Table II indicate the number of groups into which the network under consideration was divided and the values shown are derived from a simulation using a mean of 5.5 packets per message (values obtained for a mean of 10.5 packets per message are identical). The data in Table II indicates that it is very desirable to divide a network into groups and families whenever the Heritsch algorithm is employed since the same order of routing optimization is achieved with a significant reduction in the network capacity required to support the protocol.

A significant result is that for small networks the Djikstra algorithm requires less network capactiy than does the Heritsch algorithm regardless of how the network is structured for the Heritsch algorithm. While this result may appear startling given that the Heritsch algorithm purges updates from the network which are not required to be retransmitted, it must be remembered that the Heritsch algorithm also requires all "upstream" nodes to be notified whenever a new best path is adopted or a channel value is changed. This requirement causes the generation of many additional update messages the aggregate of which is

potentially larger than the total number of update messages
per cycle generated by the Djikstra algorithm. Despite the
lesser channel capacity required to implement the Djikstra
algorithm it must be remembered that the algorithm can only
be implemented synchronously which may preclude its
consideration in some applications Also for large enough
networks, the Djikstra algorithm will require more channel
utilization than would the Heritsch algorithm for a network
which is partitioned into an appropriate hairarchy of groups
and families.

3. Synchronous Versus Asynchronous Updating Performance

When the utilization versus network loading curves
for the asynchronous implementation of the Heritsch
algorithm are compared with the curves obtained for the
synchronous implementation of the algorithm, it is found
that there is very little difference in the shape and
corresponding numerical values of the respective 10/20 and
15/30 curves. While only the results for these two networks
are presented here, they are typical of results obtained
when a variety of different network topologies were tested
under different network loadings using the two
implementations of the Heritsch algorithm. The conclusion

drawn from these results is that for a given network and specified updating cycle time there is no significant advantage in implementing the Heritsch algorithm synchronously as opposed to asynchronously (or vice versa). This conclusion has considerable impact on the physical design of a packet switching network, since it is not required to precisely synchronize the adoption of new best paths throughout the network in order to obtain the full benefits of the Heritsch algorithm. This translates to a reduced complexity in the design of the packet network terminal equipment.

4. Effects Of User Message Lengths

One of the initial objectives of this study was to determine how the distribution of the lengths of user messages impacts the performance of a Distributed Routing Protocol. Since the simulation programs are constructed to operate on a predetermined packet length, the length of a user message is related to the number of packets which make up the message. In the simulations, the number of packets in each message is based upon a uniformly distributed random variable with the maximum and minimum values of the variable specified at the beginning of the simulation. The mean

value of the resulting distribution is used to control the interval between the generation of user messages. In this way the average number of packets per second entering the network is maintained as a constant for each run of the simulation, and the mean number of packets that are contained in each message can be increased arbitrarily.

The results of this investigation are plotted on each of the graphs presented in Appendix F. The graphs contain the results found when the mean number of packets in a message was 5.5 and 10.5, corresponding to maximum message lengths of 10 and 20 packet, respectively. No significant difference in the performance of either the Djikstra or the Heritsch protocols was found as a result of increasing the length of the user messages. Mean packet values of 1, 3.5 and 7.5 were also investigated but are not presented here since the results obtained do not differ substantially from those mentioned above.

B. SYNOPSIS OF RESULTS

It is not possible to conclude that a particular Distributed Routing Protocol (Djikstra or Heritsch) exhibits better overall performance characteristics than the other. Each algorithm has advantages and disadvantages which must

be considered on the basis of the network in which the protocol is to be implemented. Certainly, if optimal routing is a priority of the network, than the Djikstra algorithm is superior. However, if non-optimal routing is acceptable and complexity of terminal equipment is a consideration then the Heritsch algorithm has distinct advantages if implemented asynchronously.

The final conclusion of this study is that the length of user messages does not degrade the performance of either protocol much when the average number of packets entering the network per unit of time is relatively constant. Thus the interspersion of long and short user messages is not a factor in the design of a packet switching network.

Lastly, it appears in the context of the NTDS communications network that the Djikstra algorithm is the protocol of choice should the NTDS be converted to a packet switching network. While the asynchronous advantages of the Heritsch algorithm appear desirable for this application, the increasing amount of information the NTDS is being required to throughput, points to the adoption of a protocol which requires a minimum of network capacity to operate. Additionally, since units of the fleet are not constrained

53

by the size and complexity of the terminal equipment which is required to maintain a synchronized network (as compared to a man-pack version of a packet switching network) the choice of the Djikstra algorithm for NTDS applications is reasonable.

APPENDIX A

AVERAGE DELAY PER PACKET

The unit of measure used in this study to compare the relative performance of routing protocols in a given network is the average delay per packet (T_p). The derivation of the relation of this measure to the link utilization factors of the network is provided here and is based upon material presented in Gross and Harris [Ref. 5]. and Kleinrock [Ref. 6].

One of the basic parameters in queuing theory is the utilization factor (p) which is defined as the average arrival rate of customer (eg. packets and updates) at a service center (eg. link) multiplied by the average time required to service (eg. transmit) the customer. In general this is expressed as

$$p = lx \qquad\qquad (A-1)$$

where

      l = the average arrival rate of the customers

      x = the average time required to service a
            a customer

For this derivation the subscripts p and u are used to indicate that a particular variable is referred to packets

or updates respectively. Thus the utilization factor for packets over the i-th link is expressed as

$$p_{pi} = l_{pi} \ x_{pi} \qquad (a-2)$$

where

l_{pi} = the average number of packets per second arriving for transmission over a link.

For an M/M/1 queue, the term x_{pi} can be replaced by the term $1 / uC_i$, where u represents the percentage of a packet which is represented by 1 bit (ie. $1 / u$ represents the length of a packet in bits) and C_i is the number of bits per second which can be serviced by the i-th link. Equation (A-2) can then be rewritten as

$$p_{pi} = \frac{l_{pi}}{uC_i} \qquad (A-3)$$

Moreover, for M/M/1 systems (ie. a system with customers arriving at an exponentially distributed rate and the service time per customer is exponentially distributed) the expected time a packet spends in the i-th link's queue before servicing begins can be expressed as

56

$$W_{qpi} = \frac{l_{pi} / uc_i}{uc_i - l_{pi}} \qquad \text{(A-4)}$$

and the expected delay for the delivery of a packet over a link is

$$W_{pi} = W_{qpi} + \frac{1}{uc_i} \qquad \text{(A-5)}$$

which is to say that the expected delivery delay for a packet is the sum of the expected time the packet spent in the service queue plus the time required to service the packet.

The assumption is made for the remainder of the derivation that Equation (A-4) will provide a fair representation of the expected delay time for a packet when the service time per packet is fixed instead of probabilistic in its nature. However, since two types of messages (updates and packets) with different fixed service times are allowed to enter a link, the time available for the link to service packets is reduced by the amount of time the link is busy servicing updates. Thus Equation (A-4) must still be modified to reflect the system perturbation due to the loss of available packet service time.

Consider the case where x_{ui} is the time required to service an update message and t_{ui} is the interval between the arrival of update messages at the i-th link. The arrival rate of updates at the link (l_{ui}) is equal to $(1 / t_{ui})$. Using Equation (A-2), modified to reflect its application to update message, the utilization factor for updates becomes

$$p_{ui} = \frac{x_{ui}}{t_{ui}} \tag{A-6}$$

The fundamental interpretation of the term uC_i in Equation (A-4) is the maximum average number of packets which can be transmitted per second over the i-th link. Only a fraction $((t_{ui} - x_{ui}) / t_{ui})$ of the time is available for packets, however, when updates must also be transmitted. Accordingly, we must reduce uC_i by the factor

$$1 - \frac{x_{ui}}{t_{ui}} = 1 - p_{ui} \tag{A-7}$$

which is equivalent to reducing u to u', where

$$u' = u (1 - p_{ui}) \tag{A-8}$$

63

Replacing u with u' in Equation (A-4), a new expression for W_{pi} (Equation (A-5)) is obtained

$$W_{pi} = \left[\frac{l_{pi} / (uC_i (1 - p_{ui}))}{uC_i (1 - p_{ui}) - l_{pi}} + \frac{1}{uC_i} \right] \qquad (A-9)$$

From Kleinrock, [Ref. 6]. the average packet delay (T_p) for a network of service centers is expressed as

$$T_p = \sum_{i=1}^{k} \frac{l_{pi}}{g} \quad W_{pi} \qquad (A-10)$$

where

>   g = average total number of packets
>   per second entering the
>   network
>
>   k = number of links in the network

Substituting Equation (A-5) into equation (A-10) the expression for average delay time for a packet in the network becomes

$$T_p = \frac{1}{g} \sum_{i=1}^{k} \left[\frac{l_{pi} l_{pi} / (uC_i (1 - p_{ui}))}{uC_i (1 - p_{ui}) - l_{pi}} + \frac{l_{pi}}{uC_i} \right] \qquad (A-11)$$

After some simplification this becomes

$$T_p = -\frac{1}{g} \sum_{i=1}^{k} \left[\frac{p_{pi}^2 \ / (1 - p_{ui})}{1 - p_{ui} - p_{pi}} + p_{pi} \right] \qquad (A-12)$$

Equation (A-12) represents a method of relating the network capacity which is utilized by a routing protocol and the traffic loading level of the network to the average packet delay. For the simulation work presented in this study the utilization factors (p_{ui} and p_{pi}) were measured directly and the network loading (g) was an input parameter to the simulation.

The reader should note that Equation (A-12) is valid only for comparing the performance of protocols and not for calculation of the actual performance of a protocol.

APPENDIX B

NETWORK CONFIGURATIONS
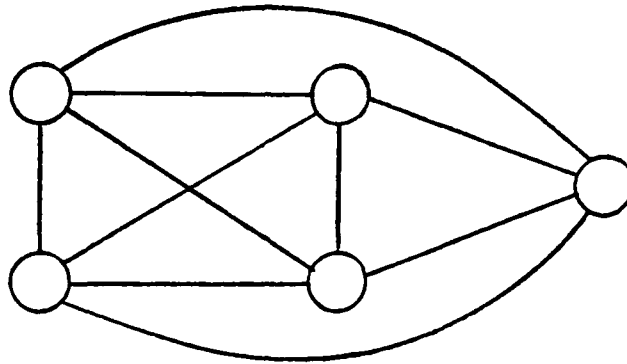
Fig. B.1.   The 5/10 Network
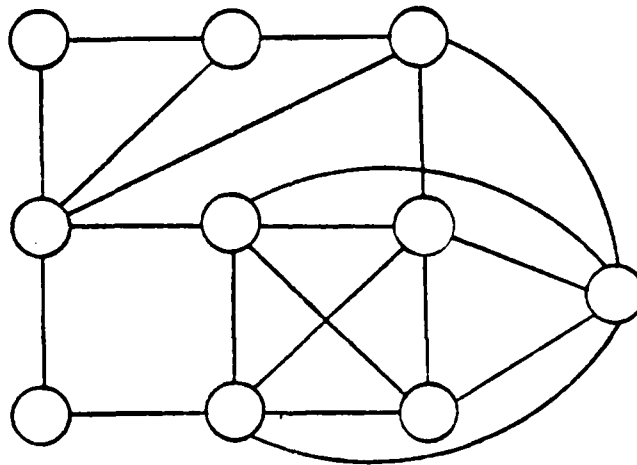
Fig. B.2.   The 10/20 Network
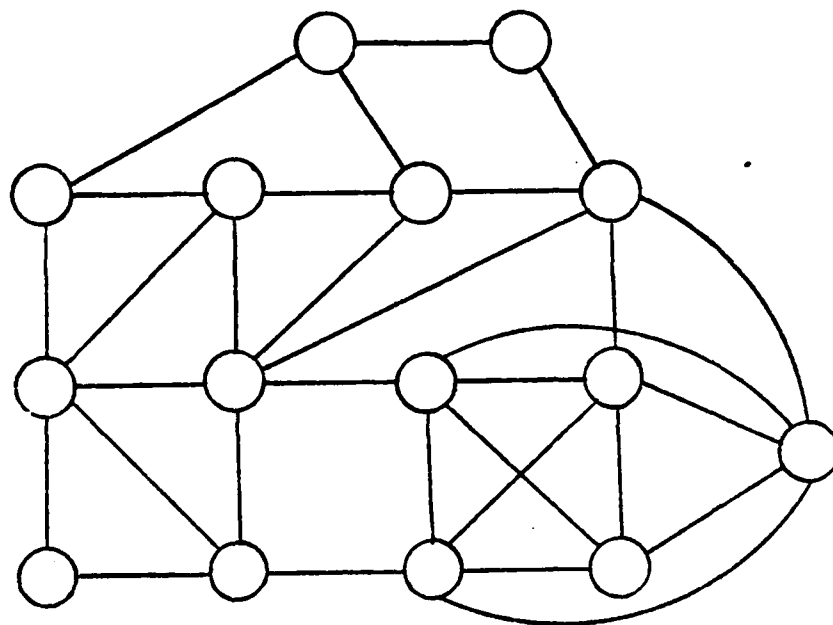
Fig. 8.3.   The 15/30 Network
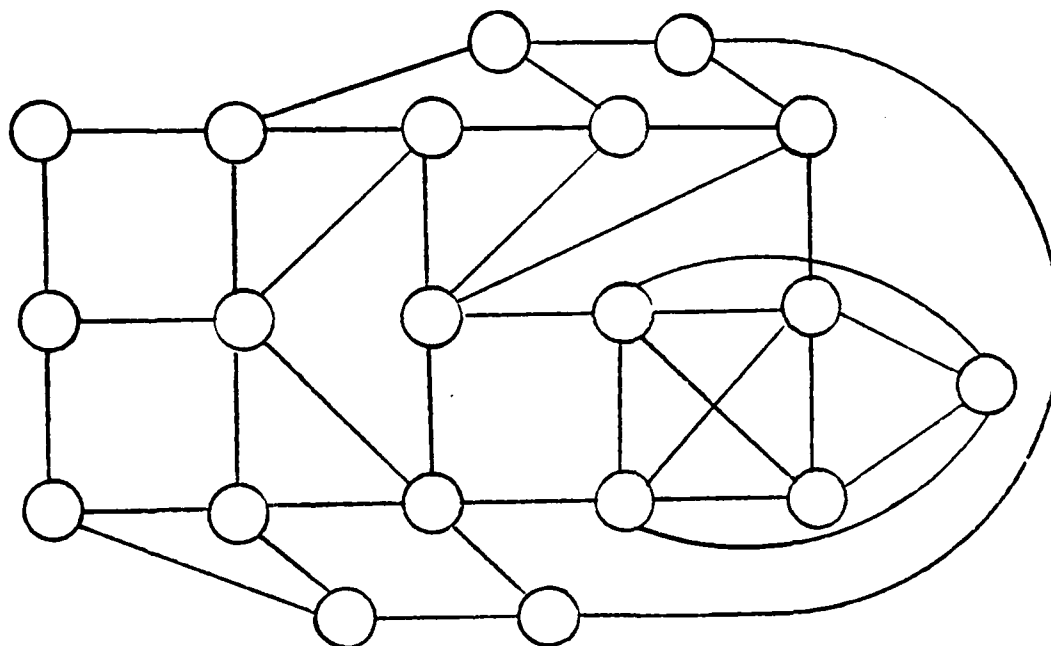
Fig. 8.4.   The 20/40 Network

72

HERITSCH ALGORITHM (SYNCHRONOUS): PROGRAM LISTING

```
//LENGERIC JCB (1611,014C),'LENGERICH 1101',CLASS=C
//*MAIN ORG=NPGVM1.161IP;LINES=(50)
//*FORMAT PR;DCNAME=DEST=LCCAL
// EXEC SIM25CLG;REGICN.GO=3072K,PARM.GO='MAP,SIZE=760K'
//SYSPRINT CD SYSCUT=A
//SIM.SYSIN CD *
PREAMBLE
''    THIS PROGRAM SIMULATES THE HERITSH DISTRIBUTED ROUTING
''         PROTOCCL (SYNCHRONOUS).
''
NORMALLY MODE IS INTEGER
GENERATE LIST ROUTINES

PERMANENT ENTITIES
EVERY NODE HAS A TRANSMIT.PERCENT, A RECEIVE.PERCENT, A GROUP,
A FAMILY, A U.TCTAL, A AVE.NUMBER.UPDATES, A ALT.AVE.NUMBER.UPDATES
AND A PKTS.RELAYED, OWNS AN UPDATE.OUT.QUEUE, A PACKET.OUT.QUEUE ,
A TIME.QUEUE AND A LNK.FLAG
   DEFINE TRANSMIT.PERCENT RECEIVE.PERCENT AVE.NUMBER.UPDATES,
   AND ALT.AVE.NUMBER.UPDATES AS REAL VARIABLES
   DEFINE UPDATE.CUT.QUEUE AND PACKET.OUT.QUEUE AS FIFO SETS
''
TEMPORARY ENTITIES
EVERY MESSAGE HAS A TYPE, A RELAYER, A NEXT.STOP, A DESTINATION,
AN INFO1, AN INFO2, AN INFO3, AN INFO4, AN INFO5, A QUEUE.ENTRY.TIME,
AN ORIG.TIME, A LINK.ENTRY.TIME AND A TOT.QUEUE.TIME
AND MAY BELONG TO A UPDATE.OUT.QUEUE, A PACKET.OUT.QUEUE,
AND A LNK.FLAG
   DEFINE ORIG.TIME AS A REAL VARIABLE
   DEFINE LINK.ENTRY.TIME AND TOT.QUEUE.TIME AS REAL VARIABLES
   DEFINE INFO4 AND QUEUE.ENTRY.TIME AS A REAL VARIABLE
EVERY PACK HAS A NUMBER, AN ENTRY.TIME, A PAC.NEIGHBOR AND MAY
BELONG TO A TIME.QUEUE
   DEFINE TIME.QUEUE AS A LIFO SET
   DEFINE ENTRY.TIME AS A REAL VARIABLE
''
EVENT NOTICES INCLUDE REPORT.RESULTS, QU.SAMPLER, SAMPLE,
COMPUTE.CV AND ADOPT.NEW.BEST.PATH,
EVERY NEW.UPDATE.MESSAGE HAS A SENDING.NODE, AND A TYPE.MESSAGE
EVERY UPDATE.ARRIVES HAS AN ID.MESSAGE.NUMBER
EVERY CONT.UPDATE.MESSAGE HAS A LAST.NODE, A NEXT.NODE, A NET.CV,
A SOURCE, A FAMILY, A HOP.CNT AND A GR.BUP
EVERY PACKET.ARRIVES HAS AN ID.NUMBER
EVERY CON.PACKET.MESSAGE HAS AN IDENT.MESSAGE.NUMBER
EVERY NEW.PACKET.MESSAGE HAS A T.MESSAGE
EVERY COMPLETED.TRIP HAS A MES.NUM
''
```

DEFINE UP.DATE.PERIOD, PROCESSING.TIME, PKT.XMN.TIME AND TIME.LIMIT AS REAL VARIABLES
DEFINE TRNS.PCNT AND RCV.PCNT AS REAL VARIABLES
DEFINE LINKS AS A VARIABLE
DEFINE MSG.HLT AS A VARIABLE
DEFINE NEIGHBOR.LIST AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE BEST.PATH AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE CHANNEL.VALUE TO MEAN INFO1
DEFINE FAM.LY TO MEAN INFO2
DEFINE GRP TO MEAN INFO5
DEFINE GRPS, FMLYS AND NGFS AS VARIABLES
DEFINE LINK.ABLE AS A 2-DIMENSIONAL ARRAY
DEFINE LINK.MONITCR AS A 3-DIMENSIONAL ARRAY
DEFINE TRACER AS A 2-DIMENSIONAL ARRAY
DEFINE CLOCK.DATA AS A 2-DIMENSIONAL REAL ARRAY
DEFINE HOP.CCUNT AS A 2-DIMENSIONAL ARRAY
DEFINE PKT.SMP.SET AS A 2-DIMENSIONAL ARRAY
DEFINE UP.SMP.SET AS A 2-DIMENSIONAL ARRAY
DEFINE PKT.QU.DISTR AS A 1-DIMENSIONAL ARRAY
DEFINE UP.QU.DISTR AS A 1-DIMENSIONAL ARRAY
DEFINE FAM.CF.GRP AS A 1-DIMENSIONAL ARRAY
DEFINE IDLE TO MEAN 0
DEFINE BUSY TC MEAN 1
DEFINE PACKET TO MEAN 2
DEFINE MSG.GENERATION.INTERVAL, MIN.PKTS.PER.MSG, MAX.PKTS.PER.MSG AS REAL VARIABLES
DEFINE EST.MAX.NR.MSGS AND NEW.MSG.TCTAL AS VARIABLES
DEFINE TRANS.NUMBER TO MEAN INFO1
DEFINE PACK.NUMBER TO MEAN INFO2
DEFINE NODES.HOPPED TO MEAN INFO3
DEFINE RELEASE.TIME TO MEAN INFO4
DEFINE FM.GP TC MEAN INFO5
DEFINE YES TC MEAN 1
DEFINE NO TC MEAN 0
DEFINE UPDATE TO MEAN 1
DEFINE TRAF.LIMIT AS A VARIABLE
DEFINE WINDCW AS A REAL VARIABLE
DEFINE UP.PAC.RATIO AS A REAL VARIABLE
DEFINE PRNT AS A VARIAELE
DEFINE IN.GROUP, IN.FAMILY, SELECTOR AS REAL VARIABLES
DEFINE NFT.U.LINKS, UP.STARTS, MAX.U.HOPS AS VARIABLES
DEFINE TI.MER AS A REAL VARIABLE
DEFINE SMP.LINKS, NO.OF.SAMPLES SMP.CNTR AS VARIABLES
DEFINE EARLIEST.UPDATE AND LATEST.UPDATE AS REAL VARIABLES
DEFINE U.XMN.TIME AS REAL VARIABLE
DEFINE PEFK AS A VARIABLE
DEFINE TOTAL.NUMBER.CYCLES AND NUMBER.CYCLES.COMPLETED AS REAL VARIABLES

```
DEFINE MAX.LINKS,PER.NODE AS VARIABLE
DEFINE TIME.ELAPSEC AND CYCLE.TIME AS REAL VARIABLES
DEFINE P.FLOW1,P.FLOW2,U.FLOW1 AND U.FLOW2 AS REAL VARIABLES
DEFINE LONGEST.PATH AS A 4-DIMENSIONAL REAL ARRAY
DEFINE LINK.TIMER AS REAL VARIABLE
DEFINE LINK.NODE.RATIC AS REAL VARIABLE, AVE.P.FLOW, AVE.U.FLOW AND
   P.FLOW.TOTAL,U.FLOW.TOTAL, AVE.P.FLOW, AVE.U.FLOW AND
   PCNT.U.FLOW AS REAL VARIABLES
DEFINE AVE.NETWORK.FLOW AS REAL VARIABLE
DEFINE SPECIFY.OUTPUT AS VARIABLE
DEFINE RERUN AS VARIABLES
DEFINE PKTS.PER.SEC.AVE AS REAL VARIABLE
DEFINE NODE.FACTOR AS A 1-DIMENSIONAL ARRAY
DEFINE HISTOGRAM AS A 1-DIMENSIONAL ARRAY
DEFINE P1.DELAY, P2.DELAY, U1.DELAY, U2.DELAY,
   TOTAL.DELAY,LINK1.FLOW, LINK2.FLOW AND
   AVE.DELAY.PER.PKT AS REAL VARIABLES
DEFINE RATIC.STRANDED AS REAL VARIABLE
DEFINE PKTS.HALF AND PKTS.END AS VARIABLE
DEFINE DEST.CCUNT AND GEN.CCUNT AS VARIABLES
DEFINE STRANDED.WAIT AND TRANSIT.TIME AS REAL VARIABLES
DEFINE T.IN.LINK AND Q.IN.LINK AS REAL VARIABLES
END ''OF PREAMBLE
''
MAIN
''
LFT LINES.V = 79
SKIP 2 OUTPUT LINES
''
PRINT 4 LINES AS FCLLOWS
######## HERITSCH UPDATE ALGORITHM: SYNCHRONOUS VERSION ########
         ## PACKET GENERATION RATE : FIXED ##
         ## INCREMENT FOR PKTS.PER.MSG : HARMONIC ##
         ## NORMALIZE: AVE.PKTS.PER.MSG.PER.NODE ##

SKIP 2 OUTPUT LINES
''
'' THE "MAIN" ROUTINE "CALLS" THE SUBROUTINE "INITIALIZATION".
'' "INITIALIZATION" SETS THE VALUE OF INPUT VARIABLES
''    WHICH WILL REMAIN CONSTANT FOR ALL RUNS OF THIS
''    SIMULATION. THIS ALLOWS THE "MAIN" ROUTINE TO ACT AS A
''    "DRIVER" ROUTINE FOR SIMULATION. THE "MAIN" CAN BE
''    STRUCTUREC TO CHANGE CERTAIN CONDITIONS OF THE SIMULATION
''    AND RERUN THE SIMULATION AGAIN.
''
DEFINE INC.PKTS.PER.MSG, LIMIT.PKTS.PER.MSG, AVE.PKTS.PER.MSG,
   PKTS.PER.SEC.MIN, PKTS.PER.SEC.INC, PKTS.PER.SEC.MAX
```

```
AS REAL VARIABLES
    SET THE MAXIMUM, INCREMENT AND MINIMUM VALUES OF THE
        VARIABLES WHICH ARE TO BE ITERATED.

    BY MULTIPLYING PKTS.PER.SEC.( ) * NODE.FACTOR THE RESULTS
        OF THE SIMULATION CAN BE NORMALIZED SO THAT NETWORKS
        WITH DIFFERENT NUMBERS OF NODES CAN BE COMPARED USING
        "AVE.PKT.PER.SEC.PER.NODE" AS A CONSTANT MEASURE OF
        NETWORK LOADING. (EG. IF A 5 NODE NETWORK IS THE
        COMPARISON BASE, THEN A 10 NODE NETWORK HAS A NODE.FACTOR
        = 2. ## NOTE: NODE.FACTORS GREATER THAN 4 REQUIRE IN
        EXCESS OF 3072K OF CORE TO RUN THE SIMULATION.

READ NODE.FACTOR

LET MIN.PKTS.PER.MSG = 10.
LET INC.PKTS.PER.MSG = 10.
LET LIMIT.PKTS.PER.MSG = 21.

LET PKTS.PER.SEC.MIN = 50.  * NODE.FACTOR
LET PKTS.PER.SEC.INC = 50.  ** NODE.FACTOR
LET PKTS.PER.SEC.MAX = 500. * NODE.FACTOR

LET PKTS.PER.SEC.AVE = PKTS.PER.SEC.MIN

    THE FOLLOWING ARRAYS WILL CONTAIN THE "X" AND "Y" VALUES OF
        DATA TO BE GRAPHED.

    "ITERATE" IS THE NUMBER OF TIMES THE VALUE OF "MAX.PKTS.PER.MSG"
        IS CHANGED AND THE SIMULATION REINITIATED.
    "POINTS" IS THE NUMBER OF TIMES THE VALUE OF "PKTS.PER.SEC.AVE"
        IS CHANGED AND THE SIMULATION RERUN. THE
        TOTAL NUMBER OF TIMES THE SIMULATION RUNS IS
        ( ITERATE * POINTS ).

LET ITERATE = 2
LET POINTS = 10

LET AVE.PKTS.PER.MSG = (1.0 + MIN.PKTS.PER.MSG) / 2.
LET MSG.GENERATION.INTERVAL = AVE.PKTS.PER.MSG / PKTS.PER.SEC.AVE

    THE VALUE OF "MAX.PKTS.PER.MSG" WILL BE INCREMENTED USING A
        HARMONIC SERIES.

LET MAX.PKTS.PER.MSG = MIN.PKTS.PER.MSG
    "RERUN" CONTROLS THE PRINTING OF THE NEIGHBOR AND CV
```

```
'          LISTING.  THE LISTING IS PRINTED ONLY FOR INITIAL
'          EXECUTION OF THE PROGRAM.
'
LET RERUN = 0
'
PERFORM INITIALIZATION
RELEASE INITIALIZATION
'
LET EST.MAX.NR.MSGS = 2 * INT.F( TIME.LIMIT / PKTS.PER.SEC.MAX , 1.0 /
                                                                 )
RESERVE TRACER ( *, * ) AS EST.MAX.NR.MSGS BY 2
'
LET RERUN = 1
'
'          THESE STATEMENTS CAUSE THE VALUES OF "ITERATE" AND
'          "POINTS" TO BE WRITTEN TO THE MASS STORAGE SYSTEM
'          FOR LATER RETRIEVAL BY A PLOTTING ROUTINE.
'
USE UNIT 8 FOR OUTPUT
WRITE ITERATE AS I 4
WRITE POINTS AS / I 4
USE UNIT 6 FOR OUTPUT
'
'          SCHEDULE INITIAL EVENTS
'
'          SCHEDULE THE FIRST UPDATE FOR EACH NODE (NEW.UPDATE.MESSAGE)
'          COMPUTE CHANNEL VALUES FOR THIS UPDATE (COMPUTE.CV)
'          SCHEDULE THE FIRST REPORT ( REPORT.RESULTS )
'          SCHEDULE THE FIRST MESSAGE GENERATION (NEW.PACKET.MESSAGE)
'          SCHEDULE A TEST FOR MAX QUEUES HALF WAY THRU RUN (QU.SAMPLER)
'
'          "TOTAL.POINTS" COUNTS THE NUMBER OF TIMES THE SIMULATION IS
'          RUN.
'
LET TOTAL.POINTS = 0
'
'DO.IT.AGAIN'
'
LET TOTAL.POINTS = TOTAL.POINTS + 1
'
FOR EACH NODE, DO
    SCHEDULE A NEW.UPDATE.MESSAGE GIVEN NODE, UPDATE IN  ( UNIFORM.F
    ( C.0, UP.DATE.PERIOD, 1 ) ) UNITS
LOOP
'
SCHEDULE A COMPUTE.CV AT 0.00
SCHEDULE A REPORT.RESULTS IN ( TIME.LIMIT / 4. ) UNITS
SCHEDULE A NEW.PACKET.MESSAGE GIVEN PACKET IN ( 2 * .1 +
```

```
EXPONENTIAL.F( MSG.GENERATION.INTERVAL, 5 ) ) UNITS
SCHEDULE A QU.SAMPLER IN ( TIME.LIMIT / 2. ) UNITS
''
START SIMULATION
''      THE NEXT GROUP CF STATEMENTS ZEROIZE ALL GLOBAL VARIABLES AND
''      ARRAYS IN PREPARATION FOR A RERUN OF THE SIMULATION.
''
LET TIME.V = 0.0
LET NEW.MSG.TOTAL = 0
LET MSG.HLT = 0
LET NET.U.LINKS = 0
LET UP.STARTS = 0
LET MAX.U.HCPS = 0
LET SMP.CNTR = 0
LET PEEK = 0
LET TOTAL.NUMBER.CYCLES = 0.0
LET NUMBER.CYCLES.COMPLETED = 0.0
LET TIME.ELAPSED = 0.0
LET LONGEST.PATH = 0
LET PKTS.HALF = 0
LET PKTS.END = 0
LET DEST.COUNT = 0
LET GEN.COUNT = 0
LET TRANSIT.TIME = 0.0
LET STRANDEC.WAIT = 0.0
LET T.IN.LINK = 0.0
LET Q.IN.LINK = 0.0

FOR A = 1 TC ( 2 * LINKS), DO
FOR B = 1 TO ( 2 * LINKS ), DO
FOR C = 1 TC 6, DO
    LET LINK.MONITOR( A, B, C ) = 0
LOOP
LOOP
LOOP
FOR A = 1 TO 2, DO
FCR B = 1 TC ( N.NODE ), CO
FOR C = 1 TO ( N.NODE ), DO
FOR D = 1 TO 4, DO
    LET LINK.TIMER( A, B, C, D ) = 0.0
LOOP
LOOP
LOOP
LOOP
FOR A = 1 TO ( 10 * N.NODE ), DO
FOR B = 1 TO 2, DO
    LET HCP.CCUNT( A, B ) = 0
```

```
        LET CLOCK.DATA( A, B ) = 0
    LOOP
    FOR A = 1 TO EST.MAX.NR.MSGS, DO
    FOR B = 1 TO 2, DO
        LET TRACER( A, B ) = 0
    LOOP
    FOR A = 1 TO 1000, DO
        LET PKT.QU.DISTR(A) = 0
        LET UP.QU.DISTR(A) = 0
    LOOP
    FOR A = 1 TO SMP.LINKS, DO
    FOR B = 1 TO 2, DO
        LET PKT.SMP.SET( A, B ) = 0
        LET UP.SMP.SET( A, B ) = 0
    LOOP
    LOOP
    RELEASE BEST.PATH( *, *, * )
    RELEASE NEIGHBOR.LIST( *, *, * )
    RELEASE HISTOGRAM
    PERFORM NEIGHBOR.INITIALIZATION
    LET PKTS.PER.SEC.AVE = PKTS.PER.SEC.AVE + PKTS.PER.SEC.INC
    IF PKTS.PER.SEC.AVE <= PKTS.PER.SEC.MAX
    '' LET MSG.GENERATION.INTERVAL = AVE.PKTS.PER.MSG / PKTS.PER.SEC.AVE
    '' IF SPECIFY.OUTPUT = 0
    '' SKIP 2 OUTPUT LINES
    '' PRINT 1 LINE AS FOLLOWS
    ##################### SIMULATION RESTART #####################
    '' SKIP 1 OUTPUT LINE
    '' PRINT 1 LINE WITH PKTS.PER.SEC.AVE AS FOLLOWS
    PKTS.PER.SEC.AVE = ***.***
    '' REGARDLESS
    '' GO DO.IT.AGAIN
    ELSE
```

```
        LET PKTS.PER.SEC.AVE = PKTS.PER.SEC.MIN
        LET MAX.PKTS.PER.MSG = MAX.PKTS.PER.MSG + INC.PKTS.PER.MSG
    IF MAX.PKTS.PER.MSG <= LIMIT.PKTS.PER.MSG
        LET AVE.PKTS.PER.MSG = (MAX.PKTS.PER.MSG + 1.0) / 2
        LET MSG.GENERATION.INTERVAL = AVE.PKTS.PER.MSG / PKTS.PER.SEC.AVE
        LET INC.PKTS.PER.MSG = INC.PKTS.PER.MSG + 1.0

''      IF SPECIFY.OUTPUT = 0

''          SKIP 2 OUTPUT LINES

''          PRINT 1 LINE AS FOLLOWS
aaaaaaaaaaaaaaa         SIMULATICN RESTART        aaaaaaaaaaaaaaaaaaaaaa

''          SKIP 1 OUTPUT LINE

''          PRINT 2 LINES WITH MAX.PKTS.PER.MSG
''              AND PKTS.PER.SEC.AVE AS FCLLOWS
MAX.PKTS.PER.MSG = ***.***
PKTS.PER.SEC.AVE = ****.***
''
''          REGARDLESS

''          GO DO.IT.AGAIN
''      ELSE

''          SKIP 1 OUTPUT LINE
''          PRINT 1 LINE AS FOLLOWS
SIMULATION STOPS
''
''          STCP
''
''
END ''OF MAIN
''
''
''      THIS RCUTINE INITIALIZES ALL THE VARIABLES IN THE SIMULATION.
''      BY PROPER STRUCTURING OF THIS ROUTINE AND THE "MAIN", THE
''      SIMULATICN CAN BE MADE TO SUCCESSIVELY RERUN ITSELF USING
''      ANY NUMBER OF NEW INPUT PARAMETERS CN EACH RUN.
''
ROUTINE FOR INITIALIZATION
''
''      SPECIFY.OUTPUT IS AN INTEGER WHICH CCNTROLS THE OUTPUT FORMAT.
''          0 => ALL INPUT DATA ,THE QUARTERLY RESULTS OF THE
''              SIMULATION AND THE SPECIAL. OUTPUT SECTICN
''              ARE PRINTED
```

80

```
''      1 => CNLY THE INPUT DATA AND THE CATA SPECIFIED
''           BY THE PROGRAMMER IN "SPECIAL.OUTPUT"
''           ARE PRINTED OUT.QUARTERLY RESULTS OF
''           THE SIMULATION ARE NOT PRINTED.
''      2 => CNLY THE DATA SPECIFIED IN "SPECIAL.OUTPUT"
''           IS PRINTED OUT.

LET SPECIFY.CUTPUT = 1

''      SET SOME VARIABLES

''      SET THE RANGE OF PACKETS PER MESSAGE ( MIN.PKTS.PER.MSG
''           AND MAX.PKTS.PER.MSG )
''      SET A LIMIT CN THE NUMBER OF MESSAGES GENERATED
''           DURING A RUN OF THE SIMULATION ( TRAF.LIMIT )
''      SET AN UPDATE MSG TO PACKET PROCESSING RATIO (U.PAC.RATIO)
''      SET AN UPDATE MSG TRANSMISSION TIME (U.XMN.TIME)

LET TRAF.LIMIT =95000
LET UP.PAC.RATIO = .1
LET U.XMN.TIME = C.002

''      REAC AND PRINT INPUT DATA

READ N.NODE
''
IF SPECIFY.OUTPUT <= 1
    PRINT 2 LINES AS FOLLOWS
NODE  TRANSMIT  RECEIVE    GROUP      FAMILY
NO.   FACTOR    FACTOR     (PGM #)    (PGM #)
REGARDLESS

CREATE EVERY NCDE
FOR EVERY NCCE
    READ TRANSMIT.PERCENT(NODE), RECEIVE.PERCENT(NODE), GROUP(NODE),
    FAMILY(NCCE)

''      TRNS.PCNT AND RCV.PCNT ARE THE SUM CF TRANSMIT AND RECEIVE FACTORS.
''      GROUP NUMBERS ARE ADDEC TO N.NODE TO GET PROGRAM GROUP NUMBERS.
''      FAMILY NUMBERS ARE ADDED TO N.NODE + THE HIGHEST GROUP NUMBER.
''      TC GET THE PROGRAM FAMILY NUMBER.
''      WITHIN THE SIMULATICN GROUPS AND FAMILIES ARE HANDLED AS IF
''           THEY WERE SUPER-NCDES.  A USEFUL ANALOGY IS TO
''           ENVISION MANY SUB-NODES WITHIN A GROUP OR FAMILY SUPER-
''           NODE. ACCESS TO THE SUB-NCDES IS CCNTROLLED BY THE
''           THE SUPER-NODE'S "ADDRESS".

FOR I = 1 TC N.NOCE, DC
```

81

```
        LET TRNS.PCNT = TRNS.PCNT + TRANSMIT.PERCENT(I)
        LET RCV.PCNT = RCV.PCNT + RECEIVE.PERCENT(I)
        IF GRPS < GROUP(I)
            LET GRPS = GROUP(I)
        REGARDLESS
''
''        SET PROGRAM GRP NUM
''
        LET GROUP(I) = GROUP(I) + N.NODE
LOOP
RESERVE FAM.OF.GRP(*) AS (GRPS + N.NODE + 25)
FOR I = 1 TO N.NODE, DO
        IF FMLYS < FAMILY(I)
            LET FMLYS = FAMILY(I)
        REGARDLESS
''
''        SET PROGRAM FAM NUM
''
        LET FAMILY(I) = N.NODE + GRPS + FAMILY(I)
        LET FAM.OF.GRP (GROUP(I)) = FAMILY(I)
LOOP
LET NGFS = N.NODE + GRPS + FMLYS
IF SPECIFY.OUTPUT <= 1, DO
    FOR I = 1 TO N.NODE, DO
        PRINT 1 LINE WITH I, TRANSMIT.PERCENT(I), RECEIVE.PERCENT(I),
        (GROUP(I) - N.NODE), GROUP(I), (FAMILY(I) - N.NODE - GRPS)
        AND FAMILY(I) AS FOLLOWS
      **.***        **.**)        **(**)        **(**)
**  LOOP
    SKIP 1 OUTPUT LINE
REGARDLESS
'
READ UP.DATE.PERIOD
READ PROCESSING.TIME
READ PKT.XMN.TIME
READ TIME.LIMIT
READ WINDOW
LET CYCLE.TIME = 2 * UP.DATE.PERIOD
LET TOTAL.NUMBER.CYCLES = TIME.LIMIT / CYCLE.TIME
'
'       IN.GROUP MEANS THE PERCENTAGE OF GENERATED MESSAGES THAT WILL
'           NOT LEAVE ITS BASIC GROUP; SIMILARLY FOR IN.FAMILY.
'           NOTE: IF ALL NODES ARE SPECIFIED TO BE MEMBERS OF THE
'           SAME GROUP AND FAMILY THEN VALUES OF IN.GROUP AND IN.FAMILY
'           ARE IGNORED BY THE PROGRAM.
'
READ IN.GROUP, IN.FAMILY
```

32

```
'''  PRNT IS AN INTEGER WHICH CONTROLS THE LEVEL OF DIAGNOSTIC PRINTING.
'''  0 -> INPUT DATA + RESULTS + LISTS INITIAL NEIGHBORS
'''  1 -> C + TRACES ALL PACKETS + LISTS BEST PATHS W/ CV AT END OF RUN
'''  2 -> 1 + ANNOUNCES CHANGES IN BEST PATHS
'''  3 -> 2 + TRACES ALL ASSIGNMENT/CALCULATION OF BEST PATH CV'S
'''  4 -> ANNOUNCE ORIGINATION, RELAY AND ARRIVAL OF
'''          ALL UPDATE MESSAGES
'''  5 -> 4 + ANNOUNCES ALL BUSY/IDLE LINKS
READ PRNT
'''
'''  SMP.LINKS IS THE NUMBER OF LINKS TO BE SAMPLED
'''  NO.OF.SAMPLES IS THE EXPONENTIAL MEAN NUMBER OF SAMPLES TO BE TAKEN
'''      AT EACH OF SMP.LINKS LINKS IN THE LAST HALF OF TEST.
'''      LINKS IS THE TOTAL NUMBER OF LINKS IN THE NETWORK.
READ SMP.LINKS, NO.OF.SAMPLES
READ LINKS
READ MAX.LINKS.PER.NODE
LET LINK.NODE.RATIO = LINKS / N.NODE
'''
'''  PRINT ALL INPUT DATA AS THE OUTPUT HEADER.
'''
IF SPECIFY.OUTPUT <= 1
'''
     SKIP 1 OUTPUT LINE
     PRINT 1 LINE WITH N.NODE AS FOLLOWS
NUMBER OF NODES IS ******
     SKIP 1 OUTPUT LINE
''
     PRINT 1 LINE WITH LINKS AS FOLLOWS
NUMBER OF LINKS IS ******
     SKIP 1 OUTPUT LINE
''
     PRINT 1 LINE WITH LINK.NODE.RATIO AS FOLLOWS
RATIO OF LINKS TO NODES IS ******.******
     SKIP 1 OUTPUT LINE
''
     PRINT 1 LINE WITH TIME.LIMIT AS FOLLOWS
TIME LIMIT IS ******.****** SEC.
     SKIP 1 OUTPUT LINE
''
     PRINT 1 LINE WITH MAX.LINKS.PER.NODE AS FOLLOWS
MAX LINKS PER NODE IS ******
     SKIP 1 OUTPUT LINE
''
```

```
PRINT 1 LINE WITH TRAF.LIMIT AS FOLLOWS
MAX.NUMBER OF MESSAGES GENERATED PER RUN IS ******
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH UP.DATE.PERIOD AS FOLLOWS
UPDATE PERIOD IS **.****** SEC.
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH PROCESSING.TIME AS FOLLOWS
PACKET PROCESSING TIME IN EACH NODE IS ***.****** SEC.
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH PROCESSING.TIME * UP.PAC.RATIO AS FOLLOWS
UPDATE PROCESSING TIME IN EACH NODE IS ***.****** SEC.
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH PKT.XMN.TIME AS FOLLOWS
TIME REQUIRED TO TRANSMIT A PACKET IS ***.****** SEC.
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH U.XMN.TIME AS FOLLOWS
TIME REQUIRED TO TRANSMIT AN UPDATE IS ***.****** SEC.
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH MSG.GENERATION.INTERVAL AS FOLLOWS
MESSAGES ARE INITIATED AT AN AVERAGE INTERVAL OF **.****** SEC.
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH WINDOW AS FOLLOWS
CHANNEL VALUE CALCULATION WINDOW IS ***.****** SEC.
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH MIN.PKTS.PER.MSG AND MAX.PKTS.PER.MSG AS FOLLOWS
EACH MESSAGE VARIES FROM **. TO **. PACKETS IN LENGTH
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH IN.GROUP AS FOLLOWS
AT LEAST **% OF MESSAGES ARE TO NODES WITHIN A GROUP.
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH IN.FAMILY AS FOLLOWS
AT LEAST **% OF MESSAGES ARE TO NODES WITHIN A FAMILY
SKIP 1 OUTPUT LINE
''

PRINT 1 LINE WITH TOTAL.NUMBER.CYCLES AS FOLLOWS
TOTAL NUMBER OF UPDATE CYCLES POSSIBLE FOR THIS RUN IS *****.**
SKIP 1 OUTPUT LINE
''
```

84

```
            PRINT 1 LINE WITH CYCLE.TIME AS FOLLOWS
CYCLE TIME IS *****.*****.****** SEC.
.. SKIP 1 OUTPUT LINE

   PRINT 1 LINE WITH NODE.FACTOR AS FOLLOWS
NODE.FACTOR = ***
.. SKIP 1 OUTPUT LINE

REGARDLESS
..    ARRAYS ARE RESERVED AS FOLLOWS
..
RESERVE LINK.ABLE(*,*) AS LINKS BY 2
RESERVE LINK.MONITOR(*,*) AS (2*LINKS) BY (2*LINKS) BY 6
RESERVE HGP.CCUNT(*,*) AS ( 10 * N.NODE ) BY 2
RESERVE CLOCK.DATA(*,*) AS ( 10 * N.NODE ) BY 2
RESERVE PKT.SMP.SET(*,*) AS SMP.LINKS BY 2
RESERVE UP.SMP.SET(*,*) AS SMP.LINKS BY 2
RESERVE UP.QU.DISTR(*) AS 1000
RESERVE LINK.TIMER(*,*,*) AS 2 BY (N.NODE) BY (N.NODE) BY 4

IF SPECIFY.OUTPUT <= 1
..
   PRINT 1 LINE AS FOLLOWS
   LINKS
..
REGARDLESS
..
FOR I = 1 TO LINKS, DO
   FOR J = 1 TO 2, READ LINK.ABLE(I,J)
..
   IF SPECIFY.OUTPUT <= 1
..
**  -  **        PRINT 1 LINE WITH LINK.ABLE(I,1), LINK.ABLE(I,2) AS FOLLOWS
..
   REGARDLESS
..
LOOP
..
PERFORM NEIGHBOR.INITIALIZATION

RETURN
END ''OF INITIALIZATION
..
..   THE FOLLOWING SUBROUTINE IDENTIFIES NEIGHBORS, SETS
```

85

```
''      INITIAL CV'S, AND PRINTS THE NEIGHBOR LIST AND
''      INITIAL CV'S.

ROUTINE FOR NEIGHBOR.INITIALIZATION
''
RESERVE HISTOGRAM(*,*) AS 10 AS NGFS BY NGFS BY 4
RESERVE BEST.PATH(*,*,*) AS NGFS BY NGFS BY 4
''
FOR I = 1 TO NGFS DO
   LET BEST.PATH (I,I,1) = I
   LET BEST.PATH (I,I,3) = I
LOOP
RESERVE NEIGHBOR.LIST(*,*,*) AS N.NODE BY MAX.LINKS.PER.NODE BY 3
FOR I = 1 TO LINKS, DO
FOR J = 1 TO MAX.LINKS.PER.NODE, DO
   IF NEIGHBOR.LIST(LINK.ABLE(I,1),J,1) = 0
      LET NEIGHBOR.LIST(LINK.ABLE(I,1),J,1) = LINK.ABLE(I,2)
      LET NEIGHBOR.LIST(LINK.ABLE(I,1),J,3) = 1
      LET M = J
      GO NEXT.STEP
   ELSE
'NEXT.STEP'
   LOOP
FOR J = 1 TO MAX.LINKS.PER.NODE, DO
   IF NEIGHBOR.LIST(LINK.ABLE(I,2),J,1) = 0
      LET NEIGHBOR.LIST(LINK.ABLE(I,2),J,1) = LINK.ABLE(I,1)
      LET NEIGHBOR.LIST(LINK.ABLE(I,2),J,3) = 1
      GO LAST.STEP
   ELSE
'LAST.STEP'
   LET NEIGHBOR.LIST(LINK.ABLE(I,1),M,2) = YES
   LET NEIGHBOR.LIST(LINK.ABLE(I,2),J,2) = YES
LOOP
''
IF SPECIFY.OUTPUT = 0 AND RERUN = 0
   SKIP 3 OUTPUT LINES
   PRINT 1 LINE AS FOLLOWS
NEIGHBORS AND CHANNEL VALUES
   FOR I = 1 TO N.NODE, DO
      SKIP 1 OUTPUT LINE
      PRINT 1 LINE WITH I AS FOLLOWS
NODE **  NEIGHBORS AND CV
      FOR J = 1 TO MAX.LINKS.PER.NODE, DO
         PRINT 1 LINE WITH NEIGHBOR.LIST(I,J,1) AND NEIGHBOR.LIST(I,J,3)
         AS FOLLOWS
**
```

```
        LOOP
    ''  LOOP
    REGARDLESS
    ''
    ''
    RETURN
    END  ''OF NEIGHBOR.INITIALIZATION
    ''  THIS ROUTINE HALTS THE PROGRAM AND GIVES SEVERAL STATISTICAL
    ''  REPORTS ON THE STATUS OF THE SIMULATION. AFTER FOUR REPORTS
    ''  THE ROUTINE RETURNS CONTROL OF THE PROGRAM TO THE "MAIN".
    ''
    EVENT REPORT.RESULTS
    DEFINE TOT.HOPS, TOT.PACKETS, TOT.HALF.PKTS, DELIVERED
           AND DLVRD.AT.HALF AS VARIABLES
           AND AVE.TIME AND NODES.HOPPED AS REAL VARIABLES
    DEFINE RATIO AND IDEAL.TIME AS REAL VARIABLES
    DEFINE P.Y AND U.Y AS REAL VARIABLES
    DEFINE U.SD,P.SD,U.XR,P.XR,U.Z2,P.Z2,U.Z2SUM,P.Z2SUM AS REAL VARIABLES
    ''
    ''  PEEK COUNT THE FOUR REPORTS
    ''  TOT.PACKETS SUMS THE TOTAL PKTS GENERATED UP TO THIS POINT
    ''  DELIVERED SUMS THE TOTAL PKTS REACHING THEIR DESTINATION
    ''  TOT.HOPS SUMS ALL HOPS MADE BY ALL PKTS
    ''  U.SUM AND P.SUM ARE THE SUMS OF THE SAMPLED Q SIZES
    ''  U.X AND P.X ARE THE TOTAL NUMBER OF SAMPLES
    ''  U.Z2SUM AND P.Z2SUM ARE THE SUMS OF THE SAMPLE SIZES SQUARED
    ''
    LET PEEK=PEEK+1
    LET TOT.PACKETS = 0
    LET DELIVERED = 0
    LET TOT.HOPS = 0
    LET P.SUM = 0
    LET U.SUM = 0
    LET P.X = 0
    LET U.X = 0
    LET P.Z2SUM = 0.0
    LET U.Z2SUM = 0.0
    ''
    ''  COMPUTE THE NUMBER OF UPDATE CYCLES WHICH HAVE BEEN COMPLETED
    ''   AT EACH REPORT INTERVAL.
    ''
    IF PEEK = 1
    LET TIME.ELAPSED = TIME.LIMIT / 4
    LET NUMBER.CYCLES.COMPLETED = TIME.ELAPSED / CYCLE.TIME
    REGARDLESS
```

```
IF PEEK = 2
   LET TIME.ELAPSED = TIME.LIMIT / 2
   LET NUMBER.CYCLES.COMPLETED = TIME.ELAPSED / CYCLE.TIME
REGARDLESS
IF PEEK = 3
   LET TIME.ELAPSED = ( 3 * TIME.LIMIT ) / 4
   LET NUMBER.CYCLES.COMPLETED = TIME.ELAPSED / CYCLE.TIME
REGARDLESS
IF PEEK = 4
   LET TIME.ELAPSED = TIME.LIMIT
   LET NUMBER.CYCLES.COMPLETED = TIME.ELAPSED / CYCLE.TIME
REGARDLESS
''  PRINT BP NEIGHBORS AND CV
''
IF SPECIFY.OUTPUT = 0
''
IF PRNT > 0
   SKIP 3 OUTPUT LINES
   FOR I = 1 TO N.NODE DO
      SKIP 1 OUTPUT LINE
      PRINT 1 LINE WITH I AS FOLLOWS
BEST PATHS FROM NODE ** TO -- DESTINATION - BP.NEIGHBOR - CV FRM BP.NODE
      FOR J = 1 TO N.NODE DO
         IF (GROUP(I) = GROUP(J) AND I NE J)
            PRINT 1 LINE WITH J,BEST.PATH(I,J,1), BEST.PATH(I,J,2)     **
            AS FOLLOWS                                                 ***
         REGARDLESS
      LOOP
      FOR J = (N.NODE+1) TO NGFS, DO
         IF (GROUP(I) NE J AND FAMILY(I) NE J AND J > N.NODE)
            IF BEST.PATH (I,J,1) NE 0
               IF (FAM.CF.GRP (J) NE FAMILY(I) AND J <= (N.NODE + GRPS))
                  GO OMIT.PRINT
               ELSE
                  PRINT 1 LINE WITH J, BEST.PATH(I,J,1), BEST.PATH(I,J,2)   **
                  AS FOLLOWS                                                ***
'OMIT.PRINT'
               REGARDLESS
            REGARDLESS
         LOOP
      REGARDLESS
''    LOOP
REGARDLESS
''
```

```
''    CCUNT PKTS CREATEC AND DELIVERED
''
FOR I = 1 TC NEW.MSG.TOTAL DO
   LET TOT.PACKETS = TOT.PACKETS + TRACER (I,1)
   LET DELIVERED = CELIVERED + TRACER (I,2)
LOOP
''
''    COMPUTE THE AVERAGE NUMBER OF UPDATES GENERATED BY EACH NODE
''              PER UPDATE CYCLE PER DESTINATICN.
''
FOR J = 1 TC N.NOCE DC
   LET AVE.NUMBER.UPDATES(J) = (( U.TOTAL(J) - NUMBER.CYCLES.COMPLETED)
        / ( N.NODE - 1 ) * ( 1 / NUMBER.UPDATES(J) * NUMBER.CYCLES.COMPLETED )
LET ALT.AVE.NUMBER.UPDATES(J) = AVE.NUMBER.UPDATES(J) *
                             ( I / ( LINKS - 1 ) )
LOOP
''
''    PRINT PKT STATISTICS
''
IF SPECIFY.CUTPUT = 0
''
BEGIN REPCRT CN A NEW PAGE
PRINT 1 LINE WITH TIME.ELAPSED AS FOLLOWS
TIME ELAPSEC IS *****.***** SEC.
SKIP 1 OUTPUT LINE
''
PRINT 1 LINE WITH NUMBER.CYCLES.COMPLETED AS FOLLOWS
NUMBER OF CYCLES CCMPLETED IS *****.**
SKIP 1 OUTPUT LINE
''
PRINT 2 LINES AS FCLLOWS
NODES       NO.      MEAN TIME       PEAK TIME       IDEAL
HOPPED     PKTS      FER PKT          TIME           TIME
FOR I = 1 TO ( 10 * N.NODE ), DO
   IF HOP.CCUNT(I,PACKET) NE 0
      LET TCT.HOPS = TCT.HOPS + (I * HOP.COUNT(I,PACKET))
      LET AVE.TIME = CLOCK.CATA(I,1) / REAL.F(HOP.COUNT(I,PACKET))
      LET ICEAL.TIME = I*PKT.XMN.TIME + (I-1)*PROCESSING.TIME
      PRINT 1 LINE WITH I, HCP.COUNT(I,PACKET), AVE.TIME,
        CLOCK.CATA(I,2) AND IDEAL.TIME AS FOLLOWS
***  ******       **    ***.******      *.******       *.******
   REGARDLESS
LGCP
''
''    PRINT ALERT MSG IF A PKT HOPPED MORE THAN TOTAL NUMBER OF NODES
''
IF MSG.HLT NE 0
   SKIP 2 CUTPUT LINES
```

89

```
         PRINT 1 LINE AS FOLLOWS
====  NOTE AT LEAST 1 PACKET HOPPED MORE THAN THE TOTAL NUMBER OF NODES =
         SKIP 2 CUTPUT LINES
REGARDLESS
''
      IF DELIVERED NE 0
         LET AVE.NODES.HOPPED = REAL.F(TOT.HOPS) / REAL.F(DELIVERED)
      ELSE
         SKIP 1 CUTPUT LINE
         PRINT 1 LINE AS FOLLOWS
..NO PACKETS HAVE BEEN DELIVERED DURING THIS PART OF THE SIMULATION..
REGARDLESS
''
''      PRINT SELECTED STATISTICAL DATA
''

      SKIP 3 OUTPUT LINES
      PRINT 1 LINE WITH AVE.NODES.HOPPED AS FOLLOWS
MEAN NUMBER OF NODES HOPPED PER PACKET IS **.*

      SKIP 1 OUTPUT LINE
      PRINT 1 LINE WITH NEW.MSG.TOTAL AND TOT.PACKETS AS FOLLOWS
A, TOTAL OF  **** NEW XMNS WERE STARTED (TOTALING ***** PACKETS ).
''
      PRINT 1 LINE WITH (TOT.PACKETS - DELIVERED) AS FOLLOWS
OF THESE,  *** PACKETS WERE UNDELIVERED WHEN THE TEST WAS ENDED.
''
      SKIP 1 OUTPUT LINE
      LET RATIO = REAL.F(NET.U.LINKS)/REAL.F(UP.STARTS)
      PRINT 1 LINE WITH RATIO AS FOLLOWS
FOR EACH NEW UPDATE, AN AVERAGE OF ***.* LINKS WERE USED.
''
      SKIP 1 OUTPUT LINE
      PRINT 1 LINE WITH LONGEST.PATH AS FOLLOWS
LONGEST PATH TAKEN BY A PACKET WAS **** LINKS.
''
      SKIP 2 OUTPUT LINES
      PRINT 4 LINES AS FOLLOWS
NODE    AVE NUMBER OF          AVE NUMBER OF        NUMBER OF           NUMBER OF
        UPDATES PER            UPDATES PER          UPDATES             PACKETS
        DESTINATION PER        DESTINATION PER      ORIGINATED          RELAYED
        CYCLE                  CYCLE PER LINK        OR RELAYED
''
      SKIP 1 OUTPUT LINE
      FOR J = 1 TO N.NODE, DO
      PRINT 1 LINE WITH J, AVE.NUMBER.UPDATES(J), ALT.AVE.NUMBER.UPDATES(J),
              .U.TOTAL(J), PKTS.RELAYED(J) AS FOLLOWS
 ***   ****.*.*****     *****.*****       *******            *******
LOOP
```

90

```
''
REGARDLESS
''
IF PEEK = 2
   LET DLVRD.AT.HALF = DELIVERED
   LET TOT.HALF.PACKETS = TOT.PACKETS
REGARDLESS
''
IF PEEK = 4
   LET RATIO.STRANDED = REAL.F(( TOT.PACKETS - TOT.HALF.PACKETS) -
                        (DELIVERED - DLVRD.AT.HALF)) / REAL.F( DELIVERED
                        - DLVRD.AT.HALF)
REGARDLESS
''
IF PEEK =2 OR PEEK = 4
''
''   CALCULATE DIRECTIONAL FLOW OF PACKETS AND UPDATES IN EACH LINK.
''   P.FLOW1 IS % TIME LINK IS BUSY PASSING PACKETS IN ONE DIRECTION
''   P.FLOW2 IS % TIME LINK IS BUSY PASSING PACKETS IN OTHER DIRECTION
''   ( SIMILARLY FOR U.FLOW1 AND U.FLOW2 )
''
   PERFORM FLOW.AND.DELAY
REGARDLESS
''
END ''OF BEGIN REPORT
''
IF SPECIFY.OUTPUT = 0
''
''   PRINT MAX UPDATE AND PACKET QUEUE LENGTHS
''      AT THE END OF EACH QUARTER
''
IF PRNT >= 0
   BEGIN REPORT ON A NEW PAGE
   PRINT 2 LINES AS FOLLOWS
MAXIMUM QUEUE LENGTHS:
                 PACKET QUEUE MAX      UPDATE QUEUE MAX
   FROM TO LINKS: CO
   FOR I = 1 TO LINKS:
      LET A = LINK.ABLE(I,1)
      LET B = LINK.ABLE(I,2)
      PRINT 1 LINE WITH A, B, LINK.MONITOR(A,B,3), LINK.MONITOR(A,B,5)
         AS FOLLOWS
***  ****   *****        ******
      PRINT 1 LINE WITH B, A, LINK.MONITOR(B,A,3), LINK.MONITOR(B,A,5)
         AS FOLLOWS
***  ****   *****        ******
   LOOP
''
END ''OF BEGIN REPORT
```

```
''  REGARDLESS
    REGARDLESS
''
''    PRINT THE QUEUE SIZES FROM 0 TO 500 AND THE NUMBER
''       OF TIMES THE UPDATE OR PACKET OUTGOING QUEUES
''       WERE FOUND TO BE OF THAT SIZE WHEN SAMPLED
''       ( IE. QUEUE SIZE SAMPLE DENSITY ).

IF SPECIFY.OUTPUT = 0
    IF PRNT >= 0
    IF PEEK >= 3
''
    BEGIN REPORT ON A NEW PAGE
    PRINT 3 LINES WITH SMP.LINKS AND SMP.CNTR AS FOLLOWS
OUTGOING LINKS WERE SAMPLED
***  SAMPLES PER LINK WERE TAKEN
    SKIP 2 OUTPUT LINES
    PRINT 2 LINES AS FOLLOWS
QUEUE       PACKET QUEUE          UPDATE QUEUE
SIZE        SAMPLE DENSITY        SAMPLE DENSITY
    FOR I = 1 TO 100C, DO
    LET P.SUM = ((I-1) ** PKT.QU.DISTR(I) + P.SUM
    LET U.SUM = ((I-1) ** UP.QU.DISTR(I) + U.SUM
    LET P.X = P.X + PKT.QU.DISTR(I)
    LET U.X = U.X + UP.QU.DISTR(I)
    LET P.ZZ = PKT.QU.DISTR(I) * ((I-1)**2)
    LET U.ZZ = UP.QU.DISTR(I) * ((I-1)**2)
    LET P.ZZSUM = P.ZZSUM + P.ZZ
    LET U.ZZSUM = U.ZZSUM + U.ZZ
    IF PKT.QU.DISTR(I) NE 0 OR UP.QU.DISTR(I) NE 0
    PRINT 1 LINE WITH (I-1), PKT.QU.DISTR(I) AND
       UP.QU.DISTR(I) AS FOLLOWS
    ****                           ****
    ****
*** REGARDLESS
    LOOP
    LET P.Y = P.SUM / P.X
    LET U.Y = U.SUM / U.X
    SKIP 1 OUTPUT LINE
    PRINT 2 LINES WITH P.Y AND U.Y AS FOLLOWS
AVE PACKET QUEUE LENGTH = ***.***
AVE UPDATE QUEUE LENGTH = ***.***
    LET P.XR = P.X
    LET U.XR = U.X
    LET P.SD = SQRT.F(( P.ZZSUM - P.XR *(P.Y**2)) / (P.XR-1.))
    LET U.SD = SQRT.F(( U.ZZSUM - U.XR *(U.Y**2)) / (U.XR-1.))
    SKIP 1 OUTPUT LINE
```

92

```
                  PRINT 2 LINES WITH P.SD AND U.SD AS FOLLOWS
PACKET QUEUE STANDARD DEVIATION = **.****
UPDATE QUEUE STANDARD DEVIATION = **.****
''    CHECK ALL UNDELIVERED PKTS.    REPORT ANY WITH A HOP COUNT > N.NODE
''
                  PRINT 1 LINE AS FOLLOWS
UNUSUAL DELAYS FOR PACKETS NOT DELIVERED DESCRIBED BELOW
FOR NC.DE = 1 TO N.NODE, DO
      FOR EACH MESSAGE IN PACKET.OUT.QUEUE(NO.DE) WITH TYPE(MESSAGE)
         = PACKET, DO
         IF NODES.HOPPED(MESSAGE) >= N.NODE
            PRINT 1 LINE WITH RELEASE.TIME(MESSAGE),
                  NODES.HOPPED(MESSAGE) AS FOLLOWS
PACKET RELEASED AT **.**.**** MAKES *** HOPS
      REGARDLESS
      LOOP

''    END ''OF BEGIN REPORT

''    REGARDLESS
REGARDLESS
REGARDLESS

IF PEEK NE 4
''
''    SCHEDULE A REPORT.RESULTS IN ( TIME.LIMIT / 4. ) UNITS

ELSE
''
      PERFORM SPECIAL.OUTPUT
      PERFORM DESTRUCTION
''
REGARDLESS
''
''
RETURN
END ''OF REPORT.RESULTS
''
''
''    THIS SUBROUTINE CALCULATES THE NETWORK FLOW AND
''         THE AVERAGE DELAY PER PACKET
''
''
ROUTINE FOR FLOW.AND.DELAY

DEFINE T.DELAY1, T.DELAY2 AND GAMMA.INVERSE AS REAL VARIABLES
DEFINE INSTANCE AND NEW.INSTANCE AS VARIABLE
```

93

```
..
LET P.FLOW.TOTAL = 0.0
LET U.FLOW.TOTAL = 0.0
LET TOTAL.DELAY = 0.0
..
IF SPECIFY.OUTPUT = 1 AND PEEK = 4
   SKIP 4 OUTPUT LINES
   PRINT 1 LINE AS FOLLOWS
##############              NEW SIMULATION              ##############
   SKIP 1 OUTPUT LINE
   PRINT 2 LINES WITH MAX.PKTS.PER.MSG AND PKTS.PER.SEC.AVE / NODE.FACTOR
        AS FOLLOWS
         MAX.PKTS.PER.MSG = **.**
         PKTS.PER.SEC.AVE = ****.**
REGARDLESS
..
''    RECORD THE LINK AND QUEUE STATISTICS AT THE HALF WAY POINT
''    OF THE SIMULATION.
..
IF PEEK = 2
   FOR I = 1 TO LINKS, DO
   LET A = LINK.ABLE(I,1)
   LET B = LINK.ABLE(I,2)
   LET LINK.TIMER( 1, A, B, 4 ) = LINK.TIMER( 1, A, B, 3 )
   LET LINK.TIMER( 1, B, A, 4 ) = LINK.TIMER( 1, B, A, 3 )
   LET LINK.TIMER( 2, A, B, 4 ) = LINK.TIMER( 2, A, B, 3 )
   LET LINK.TIMER( 2, B, A, 4 ) = LINK.TIMER( 2, B, A, 3 )
..
   LET LINK.MONITOR( A, B, 6 ) = LINK.MONITOR( A, B, 2 )
   LET LINK.MONITOR( B, A, 6 ) = LINK.MONITOR( B, A, 2 )
..
   LET PKTS.HALF = PKTS.HALF + LINK.MONITOR(A,B,6)
                             + LINK.MONITOR(B,A,6)
..
''    IF THE LINK IS BUSY AT HALF TIME , THEN ADD THE TRANSMISSION TIME
''    OF A PACKET TO THE TOTAL TIME THE LINK WAS BUSY DURING THE FIRST
''    HALF OF THE SIMULATION.
..
   IF LINK.MONITOR(A,B,1) = BUSY
      LET LINK.TIMER(1,A,B,4) = LINK.TIMER(1,A,B,4) + PKT.XMN.TIME
   REGARDLESS
   IF LINK.MONITOR(B,A,1) = BUSY
      LET LINK.TIMER(1,B,A,4) = LINK.TIMER(1,B,A,4) + PKT.XMN.TIME
   REGARDLESS
..    LOOP
REGARDLESS
..
```

```
IF PEEK = 4
   SKIP 2 OUTPUT LINES
   PRINT 3 LINES AS FOLLOWS
                -- UTILIZATION FACTOR --
   LINK                                      PKTS IN    PKTS IN    WAIT TIME
         PACKET    UPDATE    TOTAL           QUEUE AT   QUEUE AT   FOR PKTS
                                             HALF       END
'' SKIP 1 OUTPUT LINE
   FOR I = 1 TO LINKS, DO
''    LET A = LINK.ABLE(I,1)
      LET B = LINK.ABLE(I,2)
''    LET P.FLOW1 = (LINK.TIMER(1, A, B, 3) - LINK.TIMER(1, A, B, 4))
                   / (TIME.ELAPSED - (TIME.LIMIT * 0.5))
      LET P.FLOW2 = (LINK.TIMER(1, B, A, 3) - LINK.TIMER(1, B, A, 4))
                   / (TIME.ELAPSED - (TIME.LIMIT * 0.5))
      LET P.FLOW.TOTAL = P.FLOW1 + P.FLOW2 + P.FLOW.TOTAL
''    LET U.FLOW1 = (LINK.TIMER(2, A, B, 3) - LINK.TIMER(2, A, B, 4))
                   / (TIME.ELAPSED - (TIME.LIMIT * 0.5))
      LET U.FLOW2 = (LINK.TIMER(2, B, A, 3) - LINK.TIMER(2, B, A, 4))
                   / (TIME.ELAPSED - (TIME.LIMIT * 0.5))
      LET U.FLOW.TOTAL = U.FLOW1 + U.FLOW2 + U.FLOW.TOTAL
''    DEFINE N AS A REAL VARIABLE
''    LET T.DELAY1 = ( ( P.FLOW1**2 / (1.0 - U.FLOW1) ) / ( 1.0 -
                       U.FLOW1 - P.FLOW1 ) + P.FLOW1 ) / ( 1.0 -
      LET T.DELAY2 = ( ( P.FLOW2**2 / (1.0 - U.FLOW2) ) / ( 1.0 -
                       U.FLOW2 - P.FLOW2 ) + P.FLOW2 ) / ( 1.0 -
''    LET TOTAL.DELAY = TOTAL.DELAY + T.DELAY1 + T.DELAY2
''    LET LINK1.FLOW = P.FLOW1 + U.FLOW1
      LET LINK2.FLOW = P.FLOW2 + U.FLOW2
''    LET K = 1
      FOR N = 0.0 TO 1.0 BY 0.1, DO
         IF LINK1.FLOW > N AND LINK1.FLOW <= ( N + 0.1 )
            LET HISTOGRAM (K) = HISTOGRAM (K) + 1
         REGARDLESS
         IF LINK2.FLOW > N AND LINK2.FLOW <= ( N + 0.1 )
            LET HISTOGRAM (K) = HISTOGRAM (K) + 1
         REGARDLESS
''
```
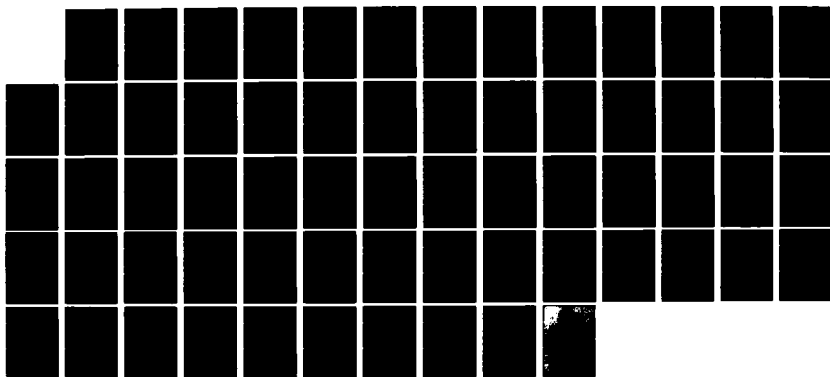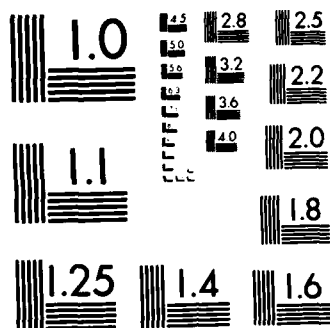
95

MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

```
IF PEEK = 4
..
  SKIP 2 OUTPUT LINES
  PRINT 3 LINES AS FOLLOWS
          -- UTILIZATICN FACTOR --     PKTS IN    PKTS IN    WAIT TIME
                                        QUEUE AT   QUEUE AT   FOR PKTS
LINK   PACKET   UPDATE   TOTAL          HALF       END AT
..
  SKIP 1 OUTPUT LINE
..
  FCR I = 1 TC LINKS, CO
..
     LET A = LINK.ABLE(I,1)
     LET B = LINK.ABLE(I,2)
..
     LET P.FLOW1 = (LINK.TIMER(1, A, B, 3) -LINK.TIMER(1, A, B, 4))
                  / (TIME.ELAPSED - (LINK.TIME.LIMIT*0.5))
     LET P.FLCW2 = (LINK.TIMER(1, B, A, 3) -LINK.TIMER(1, B, A, 4))
                  / (TIME.ELAPSED - (LINK.TIME.LIMIT*0.5))
     LET P.FLCW.TOTAL = P.FLCW1 + P.FLCW2 + P.FLOW.TOTAL
..
     LET U.FLOW1 = (LINK.TIMER(2, A, B, 3) -LINK.TIMER(2, A, B, 4))
                  / (TIME.ELAPSED - (LINK.TIME.LIMIT*0.5))
     LET U.FLCW2 = (LINK.TIMER(2, B, A, 3) -LINK.TIMER(2, B, A, 4))
                  / (TIME.ELAPSED - (LINK.TIME.LIMIT*0.5))
     LET U.FLCW.TOTAL = U.FLCW1 + U.FLOW2 + U.FLOW.TOTAL
..
  DEFINE N AS A REAL VARIABLE
..
     LET T.CELAY1 = ( ( P.FLCW1**2 / (1.0 - U.FLOW1) ) / ( 1.0 -
                     U.FLOW1 ) + P.FLOW1 ) / ( P.FLOW1 )
     LET T.CELAY2 = ( ( P.FLOW2**2 / (1.0 - U.FLOW2) ) / ( 1.0 -
                     U.FLOW2 ) + P.FLOW2 ) / ( P.FLOW2 )
..
     LET TOTAL.DELAY = TOTAL.DELAY + T.DELAY1 + T.DELAY2
..
     LET LINK1.FLOW = P.FLOW1 + U.FLOW1
     LET LINK2.FLOW = P.FLOW2 + U.FLOW2
..
     LET K = 1
     FOR N = 0.0 TC 1.0 BY 0.1, DO
        IF LINK1.FLOW > N AND LINK1.FLOW <= ( N + 0.1 )
          LET HISTCGRAM (K) = HISTOGRAM (K) + 1
        REGARDLESS
        IF LINK2.FLCW > N ANC LINK2.FLOW <= ( N + 0.1 )
          LET HISTOGRAM (K) = HISTOGRAM (K) + 1
        REGARDLESS
..
```

95

```
      LET K = K + 1
  LOOP
''
  PRINT 1 LINE WITH LINK.ABLE(I,1), LINK.ABLE (I,1), P.FLOW1, U.FLOW1,
      LINK1.FLOW, LINK.MCNITOR(A, B, 6), LINK.MONITOR(A, B, 2)
      ANC T.CELAY1 AS FOLLCWS
****/*** *.****** *.****** ****** ***** ***** *****.****** ******.******
  PRINT 1 LINE WITH LINK.ABLE (I,2), LINK.ABLE (I,2), P.FLOW2, U.FLOW2,
      LINK2.FLOW, LINK.MCNITOR(B, A, 6), LINK.MONITOR(B, A, 2)
      ANC T.DELAY2 AS FCLLCWS
****/*** *.****** *.****** ****** ****** *****.****** ******.******
''
      LET PKTS.ENC = PKTS.END + LINK.MCNITOR(A,B,2)
                             + LINK.MONITOR(B,A,2)
  LCCP
''
  LET GAMMA.INVERSE = 1.0 / PKTS.PER.SEC.AVE
  LET AVE.CELAY.PER.PKT = GAMMA.INVERSE * TOTAL.DELAY
  LET AVE.P.FLOW = P.FLOW.TCTAL / ( 2 ** LINKS )
  LET AVE.U.FLCW = U.FLCW.TOTAL / ( 2 ** LINKS )
  LET AVE.NETWORK.FLOW = AVE.P.FLOW + AVE.U.FLOW
''
  FOR EACH NCDE.CC
  FOR EACH INSTANCE IN PACKET.OUT.CUEUE(NODE), DO
      LET STRANDEC.WAIT = STRANDED.WAIT + TIME.V
                        - QUEUE.ENTRY.TIME(INSTANCE)
  LOCP
''
''  T.IN.LINK REPRESENTS THE TOTAL TIME ALL PACKETS WHICH ARE
''      NOT IN A QUEUE AND HAVE NCT BEEN DELIVERED BY THE END OF THE
''      HAVE BEEN IN THE CURRENT LINK
''  Q.IN.LINK REPRESENTS THE TOTAL TIME PACKETS WHICH ARE IN
''      CUEUE AT THE END OF THE SIMULATION HAVE BEEN IN QUEUE.
''
  FOR EACH NEW.INSTANCE IN LNK.FLAG(NODE), DO
      LET T.IN.LINK = T.IN.LINK + TIME.V - LINK.ENTRY.TIME(NEW.INSTANCE)
      LET Q.IN.LINK = Q.IN.LINK + TOT.QUEUE.TIME(NEW.INSTANCE)
  LCCP
''
REGARDLESS
''
RETURN
END ''OF FLCW.AND.CELAY
''
```

```
''    THIS SUBROUTINE CONTAINS THE SPECIAL OUTPUT FORMAT SPECIFIED
''           BY THE PROGRAMMER.  THE SUBROUTINE IS ONLY EXECUTED
''           WHEN THE SIMULATION HAS COMPLETED ALL FOUR QUARTERS
''           OF ITS RUN AND SPECIFY.OUTPUT = 1.

ROUTINE FCR SPECIAL.OUTPUT
''
DEFINE N AND CHECK.DELAY AS A REAL VARIABLES
''
SKIP 1 OUTPUT LINE
PRINT 3 LINES WITH AVE.P.FLOW, AVE.U.FLOW ANC AVE.NETWORK.FLOW
      AS FOLLOWS
AVE.P.FLOW = **.*******
AVE.U.FLOW = **.*******
AVE.NETWORK.FLOW = **.*******
''
LET CHECK.DELAY = (TRANSIT.TIME + STRANDED.WAIT + T.IN.LINK +
                  Q.IN.LINK) / REAL.F(GEN.CCUNT) DEST.CCUNT, TRANSIT.TIME,

PRINT 4 LINES WITH GEN.CCUNT, DEST.CCUNT AS FOLLOWS
      STRANDED.WAIT AS FOLLOWS
GEN.CCUNT = **.*********
DEST.CCUNT = **.*********
TRANSIT.TIME = *******.*******
STRANDED.WAIT = ********.*******
PRINT 3 LINES WITH T.IN.LINK AND Q.IN.LINK AND CHECK.DELAY AS FOLLOWS
T.IN.LINK = *******.*******
Q.IN.LINK = *******.*******
CHECK.DELAY = *******.*******
''
PRINT 2 LINES WITH AVE.DELAY.PER.PKT AND
      RATIO.STRANDED AS FOLLOWS
AVE.DELAY.PER.PKT = *****.*******
RATIC.STRANCED = *****.*******
''
PRINT 2 LINES WITH PKTS.HALF AND PKTS.END
      AS FOLLOWS
PKTS IN QUEUE AT HALF = *********
PKTS IN QUEUE AT END = *********
''
SKIP 1 OUTPUT LINE
PRINT 1 LINE AS FOLLOWS
INTERVAL    NUMBER CF LINKS

LET N = 0.0
FOR I = 1 TC 10, DO
  PRINT 1 LINE WITH N, ( N + .1 ) ANC HISTOGRAM( I )
      AS FOLLOWS
*.** - *.**           ****
```

97

```
''  LET N = N + .1
LOOP
''
''    THE FOLLOWING STATEMENTS CAUSE THE RESULTS OF THE
''    SIMULATION TO BE WRITTEN TO THE MASS STORAGE
''    SYSTEM FOR LATER RETRIEVAL BY A PLOTTING
''    ROUTINE.
''
USE UNIT 8 FCR OUTPUT
WRITE PKTS,PER.SEC.AVE/NCDE.FACTOR, AVE.NETWORK.FLOW,
    AVE.P.FLOW, AVE.U.FLOW, AVE.DELAY, PER.PKT, CHECK.DELAY AS
    /D(6,2),S 1, D(9,6),S 1, D(9,6),S 1, D(9,6), S 1,
    /D(9,6)
USE UNIT 6 FCR OUTPUT
''
RETURN
END ''OF SPECIAL.CUTPUT
''
''    THE FOLLOWING SUBROUTINE CANCELS AND/OR DESTROYS ALL ENTITIES
''    AND EVENTS WHICH REMAIN IN THE TIMING ROUTINE AFTER
''    TIME.V = TIME.LIMIT. SINCE THIS ACTICN EMPTIES THE
''    TIMING ROUTINE, CONTROL OF THE PROGRAM IS RETURNED TO THE
''    STATEMENT FOLLOWING 'START SIMULATION' IN THE ''MAIN''
''
ROUTINE FOR CESTRLCTION
''
DEFINE OCCURANCE AS VARIABLE
''
FOR EACH QU.SAMPLER IN EV.S(I.QU.SAMPLER), DO
    CANCEL THE QU.SAMPLER
    DESTROY THE QU.SAMPLER
LOOP
FOR EACH SAMPLE IN EV.S(I.SAMPLE), DC
    CANCEL THE SAMPLE
    DESTROY THE SAMPLE
LOOP
FOR EACH COMPUTE.CV IN EV.S(I.COMPUTE.CV), DO
    CANCEL THE COMPUTE.CV
    DESTROY THE COMPUTE.CV
LOOP
FOR EACH ADOPT.NEW.BEST.PATH IN EV.S(I.ADOPT.NEW.BEST.PATH), DO
    CANCEL THF ADOPT.NEW.BEST.PATH
    DESTROY THE ADOPT.NEW.BEST.PATH
LOOP
FOR EACH NEW.UPDATE.MESSAGE IN EV.S(I.NEW.UPDATE.MESSAGE), DO
    CANCEL THE NEW.UPDATE.MESSAGE
```

98

```
DESTROY THE NEW.UPDATE.MESSAGE
LOOP
FOR EACH UPDATE.ARRIVES IN EV.S(I.UPDATE.ARRIVES), DO
CANCEL THE UPDATE.ARRIVES
DESTROY THE UPDATE.ARRIVES
LOOP
FOR EACH CONT.UPDATE.MESSAGE IN EV.S(I.CONT.UPDATE.MESSAGE), DO
CANCEL THE CONT.UPDATE.MESSAGE
DESTROY THE CONT.UPDATE.MESSAGE
LOOP
FOR EACH PACKET.ARRIVES IN EV.S(I.PACKET.ARRIVES), DO
CANCEL THE PACKET.ARRIVES
DESTROY THE PACKET.ARRIVES
LOOP
FOR EACH CCN.PACKET.MESSAGE IN EV.S(I.CCN.PACKET.MESSAGE), DO
CANCEL THE CCN.PACKET.MESSAGE
DESTROY THE CCN.PACKET.MESSAGE
LOOP
FOR EACH NEW.PACKET.MESSAGE IN EV.S(I.NEW.PACKET.MESSAGE), DO
CANCEL THE NEW.PACKET.MESSAGE
DESTROY THE NEW.PACKET.MESSAGE
LOOP
FOR EACH COMPLETED.TRIP IN EV.S(I.COMPLETED.TRIP), DO
CANCEL THE COMPLETED.TRIP
DESTROY THE COMPLETED.TRIP
LOOP
..
FOR EACH NODE, DO
..
FOR EACH OCCURANCE OF UPDATE.OUT.QUEUE, DO
REMOVE OCCURANCE FROM UPDATE.OUT.QUEUE
DESTROY MESSAGE CALLED OCCURANCE
LOOP
FOR EACH OCCURANCE OF PACKET.OUT.QUEUE, DO
REMOVE OCCURANCE FROM PACKET.OUT.QUEUE
DESTROY MESSAGE CALLED OCCURANCE
LOOP
FOR EACH OCCURANCE OF LNK.FLAG, DO
REMOVE OCCURANCE FROM LNK.FLAG
DESTROY MESSAGE CALLED OCCURANCE
LOOP
FOR EACH OCCURANCE OF TIME.QUEUE, DO
REMOVE OCCURANCE FROM TIME.QUEUE
DESTROY PACK CALLED OCCURANCE
LOOP
..
LOOP
..
```

```
RETURN
END '' OF DESTRUCTION
''
''    THIS ROUTINE IS CALLED WHEN A NCDE ORIGINATES AN UPDATE MESSAGE.
''    THE INITIAL U-MSG IS SENT TO ALL OF THE INITIATING NODE'S NEIGHBORS
''
EVENT NEW.UPDATE.MESSAGE GIVEN SENDING.NCDE AND TYPE.MESSAGE
LET UP.STARTS = UP.STARTS + 1
FOR I = 1 TC LINKS, DO
''
''    CHECK EACH LINK TC SEE IF SENDING.NODE IS ON CNE END
''
IF (LINK.ABLE(I,1) = SENDING.NODE CR LINK.ABLE (I,2) = SENDING.NODE)
   CREATE A MESSAGE
   LET TYPE(MESSAGE) = UPDATE
   LET RELAYER(MESSAGE) = SENDING.NCDE
   LET U.TOTAL(SENDING.NODE) = U.TOTAL(SENDING.NODE) + 1
   IF LINK.ABLE(I,1) = SENDING.NODE
''
''    IF OPPCSITE NCDE IS IN ANOTHER FAMILY:
''
   IF FAMILY (LINK.ABLE(I,2)) NE FAMILY (SENDING.NODE)
      LET FAM.LY (MESSAGE) = FAMILY (SENDING.NODE)
      LET DESTINATION(MESSAGE) = FAMILY (SENDING.NODE)
      GC LIST.NEXT.STOP
   ELSE
''
''    IF OPPCSITE NCDE IS IN ANOTHER GROUP (SAME FAMILY):
''
   IF GROUP (LINK.ABLE(I,2)) NE GROUP (SENDING.NODE)
      LET GRP (MESSAGE) = GROUP (SENDING.NODE)
      LET DESTINATION (MESSAGE) = GRCUP (SENDING.NODE)
      GC INCL.NEXT.STOP
   ELSE
''
''    IF OPPCSITE NCDE IS IN SAME BASIC GRCUP
''
      LET DESTINATION(MESSAGE) = LINK.ABLE(I,1)
      LET NEXT.STOP;
'LIST.NEXT.STOP(MESSAGE) = LINK.ABLE(I,2)
   ELSE
   IF FAMILY (LINK.ABLE(I,1)) NE FAMILY (SENDING.NODE)
      LET FAM.LY (MESSAGE) = FAMILY (SENDING.NODE)
      LET DESTINATION (MESSAGE) = FAMILY (SENDING.NODE)
      GO INCL.NEXT.STOP
   ELSE
   IF GRCUP (LINK.ABLE(I,1)) NE GROUP (SENDING.NODE)
```

```
        LET GRP (MESSAGE) = GROUP (SENDING.NODE)
        LET DESTINATICN (MESSAGE) = GROUP (SENDING.NODE)
        GC INCL.NEXT.STOP
    ELSE
        LET DESTINATION(MESSAGE) = LINK.ABLE(I,2)
'INCL.NEXT.STOP'
        LET NEXT.STOP (MESSAGE) = LINK.ABLE(I,1)
    REGARDLESS
    IF PRNT >= 4
        SKIP 1 OUTPUT LINE
        PRINT 1 LINE WITH SENDING.NODE,NEXT.STOP(MESSAGE),
            DESTINATION(MESSAGE) AND TIME.V AS FOLLOWS
UPDATE ORIGINATED BY *** TO *** FOR DESTINATION *** AT ****.****** SEC
    REGARDLESS

..   IF LINK TO NEIGHBCR IS IDLE, SEND OUT UPDATE

..   IF LINK.MONITCR( RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 1 ) = IDLE

..   SCHEDULE ARRIVAL CF UPDATE AT OPPOSITE NCDE

..   SCHECULE AN UPDATE.ARRIVES GIVEN MESSAGE IN
        U.XMN.TIME UNITS
        LET LINK.TIMER( 2, RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 1 )
            = TIMF.V
        LET LINK.MONITCR( RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 1 ) = BUSY

..      IF PRNT >= 4
            SKIF 1 OUTPUT LINE
            PRINT 2 LINES WITH SENDING.NCDE AND U.TOTAL(SENDING.NODE)
                AS FCLLOWS
UPDATE DOES NOT WAIT IN QUEUE.
NODE *** HAS PASSED CR ORIGINATED ******** UPDATES
        REGARDLESS

..      IF PRNT >= 5
            SKIF 1 OUTPUT LINE
            PRINT 1 LINE WITH RELAYER(MESSAGE), NEXT.STOP(MESSAGE) AND
                TIME.V AS FOLLOWS
LINK *** / *** BUSY TRANSMITTING UPDATE AT ****.****** SEC.
        REGARDLESS

    ELSE

..      IF THE LINK IS BUSY, STCRE UPDATE AT THE END OF THE OUTGOING
..      UPDATF QUEUE FCR THAT LINK

..      FILF MESSAGE IN UPDATE.OUT.QUEUE( RELAYER(MESSAGE) )
```

101

```
        LET LINK.MONITOR(RELAYER(MESSAGE),NEXT.STOP(MESSAGE)) =
        IF LINK.MONITOR(RELAYER(MESSAGE),NEXT.STOP(MESSAGE)) > 4) + 1
        LET LINK.MONITOR(RELAYER(MESSAGE),NEXT.STOP(MESSAGE)) > 4)
            LINK.MONITOR(RELAYER(MESSAGE),NEXT.STOP(MESSAGE)) = 5)
        REGARDLESS
    ''
        IF PRNT >= 4
        SKIP 1 OUTPUT LINE
        PRINT 2 LINES WITH SENDING.NODE, TIME.V, SENDING.NODE AND
            LINK.MONITOR( RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 4 )
            AS FOLLOWS
UPDATE ENTERS OUTGOING UPDATE QUEUE AT NODE *** AT ****.****** SEC
        OUTGOING UPDATE QUEUE AT NODE *** IS *******
        REGARDLESS
    ''
    ''  REGARDLESS
    ''
    ''  REGARDLESS
    LOOP
    ''
    ''  SCHEDULE THE NEXT ORIGINATION OF A U-MSG FOR THIS NODE
    ''
SCHEDULE A NEW.UPDATE.MESSAGE GIVEN SENDING.NODE AND UPDATE AT
    (UNIFORM.F(EARLIEST.UPDATE, LATEST.UPDATE, 3))
RETURN
END ''OF NEW.UPDATE.MESSAGE
''...
''...  THIS ROUTINE CREATES CONTINUED U-MSGS REPRESENTING THE RELAYING
''...  OF A U-MSG FROM A NODE AFTER AN UPDATE
''...
EVENT CONT.UPDATE.MESSAGE GIVEN LAST.NODE, NEXT.NODE, NET.CV,
SOURCE, FA.MILY, MESSAGE, HOP.CNT AND GR.OUP
CREATE A MESSAGE
    LET TYPE(MESSAGE) = UPDATE
    LET RELAYER(MESSAGE) = LAST.NODE
    LET NEXT.STOP(MESSAGE) = NEXT.NODE
    LET DESTINATION(MESSAGE) = SOURCE
    LET CHANNEL.VALUE(MESSAGE) = NET.CV
    LET GRP(MESSAGE) = GR.OUP
    LET FAM.LY(MESSAGE) = FA.MLY
    LET NODES.HOPPED(MESSAGE) = HOP.CNT
    LET U.TOTAL(LAST.NODE) = U.TOTAL(LAST.NODE) + 1
    ''
    ''  IF LINK IS IDLE, SEND OUT UPDATE; OTHERWISE, PLACE UPDATE IN QUEUE
    ''
```

```
      IF LINK.MONITOR(RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 1) = IDLE

          SCHEDULE THE RELAYED UPDATE TO ARRIVE AT THE NEXT DESTINATION

          SCHEDULE AN UPDATE.ARRIVES GIVEN MESSAGE IN U.XMN.TIME UNITS
          LET LINK.TIMER(2, RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 1)
             = TIME.V
          LET LINK.MONITOR(RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 1)
             = BUSY

''        IF PRNT >= 4
             SKIP 1 OUTPUT LINE
             PRINT 3 LINES WITH RELAYER(MESSAGE), NEXT.STOP(MESSAGE),
                DESTINATION(MESSAGE), TIME.V, LAST.NODE AND
                U.TOTAL(LAST.NODE) AS FOLLOWS
UPDATE RELAYED FROM *** TO *** FOR DESTINATION *** AT ****.****** SEC
RELAYED UPDATE DOES NOT WAIT IN OUTGOING UPDATE QUEUE
NODE *** HAS PASSED ******** UPDATES
             REGARDLESS

''        IF PRNT >= 5
             SKIP 1 OUTPUT LINE
             PRINT 1 LINE WITH RELAYER(MESSAGE), NEXT.STOP(MESSAGE) AND
                TIME.V AS FOLLOWS
LINK *** / *** BUSY TRANSMITTING UPDATE AT ****.****** SEC.
             REGARDLESS

      ELSE

''        IF LINK IS BUSY, STORE THE UPDATE AT THE END OF THE OUTGOING UPDATE
''        QUEUE FOR THAT LINK

          FILE MESSAGE IN UPDATE.OUT.QUEUE(RELAYER(MESSAGE))
          LET LINK.MONITOR(RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 4) =
             LINK.MONITOR(RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 4) + 1
          IF LINK.MONITOR(RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 4) > 5
             LET LINK.MONITOR(RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 5) =
                LINK.MONITOR(RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 4)
          REGARDLESS

''        IF PRNT >= 4
             SKIP 1 OUTPUT LINE
             PRINT 3 LINES WITH RELAYER(MESSAGE), NEXT.STOP(MESSAGE),
                DESTINATION(MESSAGE), TIME.V, RELAYER(MESSAGE) AND
                LINK.MONITOR(RELAYER(MESSAGE), NEXT.STOP(MESSAGE), 4 )
                AS FOLLOWS
UPDATE FROM NODE *** TO *** WITH DESTINATION NODE *** ENTERS
```

```
        OUTGOING UPDATE QUEUE AT ****.****** SEC
        OUTGOING UPDATE QUEUE FROM NODE *** IS ********
        REGARDLESS
..
..      REGARDLESS
..
RETURN
END ''OF CONT.UPDATE.MESSAGE
..
..      THIS ROUTINE PROCESSES AN UPDATE MESSAGE AS IT ARRIVES IN A NODE,
..      RELAYING IT TO NEIGHBORS IF APPROPRIATE
..
EVENT UPDATE.ARRIVES GIVEN ID.MESSAGE.NUMBER
..
DEFINE RELAY.NODE AND MSG.NR AS VARIABLES
..
LET MSG.NR = IC.MESSAGE.NUMBER
..
LET LINK.TIMER( 2, RELAYER(MSG.NR), NEXT.STOP(MSG.NR), 2 ) = TIME.V
LET LINK.TIMER( 2, RELAYER(MSG.NR), NEXT.STOP(MSG.NR), 3 )
LET LINK.TIMER( 2, RELAYER(MSG.NR), NEXT.STOP(MSG.NR), 3 ) =
LET LINK.TIMER( 2, RELAYER(MSG.NR), NEXT.STOP(MSG.NR), 2 ) = + 1
LET LINK.TIMER( 2, RELAYER(MSG.NR), NEXT.STOP(MSG.NR), 1 )
..
LET THIS.NODE = NEXT.STOP(MSG.NR)
LET NET.U.LINKS = NET.U.LINKS + 1
LET NODES.HOPPED(MSG.NR) = NODES.HOPPED(MSG.NR) + 1
..
IF PRNT >= 4
    SKIP 1 OUTPUT LINE
    PRINT 1 LINE WITH NEXT.STOP(MSG.NR), RELAYER(MSG.NR),
        DESTINATION(MSG.NR) AND TIME.V AS FOLLOWS
UPDATE ARRIVES AT *** FROM *** FOR DESTINATION *** AT ****.****** SEC.
REGARDLESS
..
..      IC CV OF LAST RELAYING NODE
..
FOR I = 1 TO MAX.LINKS.PER.NODE, DO
    IF NEIGHBOR.LIST(THIS.NODE, I) = RELAYER(MSG.NR)
        LET CV.OF.LINK = NEIGHBOR.LIST(THIS.NODE, I, 3)
        GO SELECT.BEST.PATH
    ELSE
LOOP
'SELECT.BEST.PATH'
LET TOTAL.CV.OF.PATH = CV.OF.LINK + CHANNEL.VALUE(MSG.NR)
..
..      IC PREVIOUSLY SELECTED BP NEIGHBOR
```

104

```
''  LET BP.NEIGHBOR = BEST.PATH(THIS.NODE,DESTINATION(MSG.NR),3)
''
''       IF RELAYER = CURRENT BP NEIGHBOR, UPDATE ITS CV TO THE DESTINATION
''       AND RELAY UPDATE
''
IF RELAYER(MSG.NR) = BP.NEIGHBOR
''  LET BEST.PATH(THIS.NODE,DESTINATION(MSG.NR),4) = CHANNEL.VALUE(MSG.NR)

    IF PRNT >= 3
    SKIP 1 OUTPUT LINE
    PRINT 2 LINES WITH THIS.NODE, DESTINATION(MSG.NR), BP.NEIGHBOR,
        TIME.V, CHANNEL.VALUE(MSG.NR), CV.OF.LINK AND TOTAL.CV.OF.PATH
        AS FOLLOWS
NODE *** UPDATES CV THRU SAME BP TO *** (THRU ***) AT ****.****** SEC
        CV = *** + *** = ***
    SKIP 1 OUTPUT LINE
    REGARDLESS

''  GO RELAY.UPDATE.TO.NEIGHBORS
ELSE
''       IF THERE WAS NO BP NEIGHBOR, ADOPT RELAYER AS BP NEIGHBOR AND
''       RELAY UPDATE
''
IF BP.NEIGHBOR = NONE
    LET BEST.PATH(THIS.NODE,DESTINATION(MSG.NR),3) = RELAYER(MSG.NR)
    LET BEST.PATH(THIS.NODE,DESTINATION(MSG.NR),4) = CHANNEL.VALUE(MSG.NR)
    LET BP.NEIGHBOR = RELAYER(MSG.NR)

    IF PRNT >= 3
    SKIP 1 OUTPUT LINE
    PRINT 2 LINES WITH THIS.NODE, DESTINATION(MSG.NR), BP.NEIGHBOR,
        TIME.V AND TOTAL.CV.OF.PATH AS FOLLOWS
NEW BEST PATH FROM *** TO *** NOW THRU *** AT ****.****** SEC.
        BEST PATH CV = ******
    SKIP 1 OUTPUT LINE
    REGARDLESS

''  GO RELAY.UPDATE.TO.NEIGHBORS
ELSE
''       IF THE RELAYER IS NOT THE BP NEIGHBOR, AND IF THE NEW PATH IS
''       SHORTER THAN THE OLD BEST PATH, MAKE RELAYER THE NEW BP NEIGHBOR
''       AND RELAY THE UPDATE
''
FOR I = 1 TO MAX.LINKS.PER.NODE, DO
    IF NEIGHBOR.LIST(THIS.NODE,I,1) = BP.NEIGHBOR
```

105

```
        LET CV.TC.BP.NEIGHBOR = NEIGHBOR.LIST(THIS.NODE,I,3)
        GO COMPARE.CVS
    ELSE
    LOOP
    'COMPARE.CVS'
    IF (BEST.PATH(THIS.NODE,DESTINATION(MSG.NR),4) + CV.TO.BP.NEIGHBOR) >
    TOTAL.CV.OF.PATH
        LET OLD.BP = BP.NEIGHBOR
        LET OLD.CV = CV.TO.BP.NEIGHBOR
        LET LNK.CV = BEST.PATH(THIS.NODE, DESTINATION(MSG.NR), 4)
        LET BEST.PATH(THIS.NODE, DESTINATION(MSG.NR), 3) = RELAYER(MSG.NR)
        LET BEST.PATH(THIS.NODE, DESTINATION(MSG.NR), 4) = CHANNEL.VALUE
        (MSG.NR)
        LET BP.NEIGHBOR = RELAYER(MSG.NR)
        ''
    IF PRNT >= 2
        SKIP 1 OUTPUT LINE
        PRINT 2 LINES WITH THIS.NODE, DESTINATION(MSG.NR), BP.NEIGHBOR,
        TIME.V, CHANNEL.VALUE(MSG.NR), CV.OF.LINK AND
        TOTAL.CV.OF.PATH AS FOLLOWS
    NEW BEST PATH FROM **** TC **** NOW THRU *** AT ****.****** SEC.
            CV = ***** + ****
        PRINT 1 LINE WITH OLD.BP, OLD.CV, LNK.CV, (OLD.CV + LNK.CV) AS FOLLOWS
            OLD BP THRU *** HAD CV OF *** + *** = ***
        SKIP 1 OUTPUT LINE
    REGARDLESS
        ''
        GO RELAY.UPDATE.TO.NEIGHBORS
    ELSE
    '' IF NEW PATH IS NOT BETTER THAN THE OLD BEST PATH, DISCONTINUE U-MSG
        ''
        GO DISCONTINUE.ORIGINAL.MESSAGE
    '' IF A NEW BP IS SELECTED OR AN OLD BP IS UPDATED, PREPARE INFORMATION
    '' FOR THE NEXT U-MSG TO ALL NEIGHBORS
        ''
    'RELAY.UPDATE.TO.NEIGHBORS'
    FOR I = 1 TC MAX.LINKS.PER.NODE, DO
        IF NEIGHBOR.LIST(THIS.NODE, I, 1) = BP.NEIGHBOR
            LET CV.TO.BP.NEIGHBOR = NEIGHBOR.LIST (THIS.NODE, I, 3)
            GO COMPUTE.NET.CV
        ELSE
    LOOP
    'COMPUTE.NET.CV'
    LET NET.CV.FROM.THIS.NODE = BEST.PATH (THIS.NODE, DESTINATION(MSG.NR),4)
    + CV.TO.BP.NEIGHBOR
    FOR I = 1 TC MAX.LINKS.PER.NODE, DO
        IF NEIGHBOR.LIST (THIS.NODE, I, 2) = YES
```

106

```
IF NEIGHBOR.LIST (THIS.NODE, I, 1) NE RELAYER (MSG.NR)
   LET UPSTREAM.NODE = NEIGHBOR.LIST (THIS.NODE, I, 1)

   ''   IF UPSTREAM NODE IS IN ANOTHER FAMILY AND THIS IS A INTRA-FAMILY
   ''      U-MSG, RELAY IT

   ''   IF (FAM.LY(MSG.NR) NE 0 AND FAMILY(UPSTREAM.NODE) NE FAM.LY(MSG.NR))
   ''      GO RETRANS.UPSTREAM
        ELSE

   ''   IF UPSTREAM NODE IS IN ANOTHER GROUP AND THIS IS A INTRA-GROUP (SAME
   ''      FAMILY) U-MSG, RELAY IT

   ''   IF (GRP(MSG.NR) NE 0 AND GROUP(UPSTREAM.NODE) NE GRP(MSG.NR))
   ''      GO RETRANS.UPSTREAM
        ELSE

   ''   IF UPSTREAM NODE IS IN THE SAME BASIC GROUP AND THIS IS A BASIC
   ''      GROUP ORIGINATED U-MSG, RELAY IT

   ''   IF (GROUP(UPSTREAM.NODE) = GROUP(THIS.NODE)
   ''      AND FAM.LY(MSG.NR) = GRP(MSG.NR))
   ''         GO RETRANS.UPSTREAM
        ELSE

   ''   IF NONE OF THE ABOVE, UPSTREAM NODE DOES NOT GET A U-MSG

   ''      GO SKIP.PRINT
'RETRANS.UPSTREAM'
        LET HOPS = NODES.HOPPED(MSG.NR)
        SCHEDULE A CONT.UPDATE.MESSAGE GIVEN THIS.NODE, UPSTREAM.NODE,
        NET.CV.FROM.THIS.NODE, DESTINATION(MSG.NR), FAM.LY(MSG.NR), HOPS,
        AND GRP(MSG.NR) IN (PROCESSING.TIME * UP.PAC.RATIO) UNITS
'SKIP.PRINT'
        REGARDLESS
        REGARDLESS
LOOP
   ''   IN THE SIMULATION, ALL U-MSGS ARE DESTROYED AFTER TRAVELING ONE
   ''      LINK, HOWEVER THE UPDATE CYCLE PROCEEDS ACCORDING TO THE BASIC
   ''      CONCEPT BECAUSE THE ARRIVING U-MSG CAUSES NEW U-MSGS TO BE
   ''      INITIATED IF A NEW BP WAS SELECTED OR AN OLD BP WAS UPDATED.
   ''      IF A NODE COULD NOT USE AN INCOMING U-MSG, IT IS DESTROYED
   ''      WITHOUT GENERATING ANY NEW U-MSGS.

'DISCONTINUE.ORIGINAL.MESSAGE'
IF NODES.HOPPED(MSG.NR) > MAX.U.HOPS
   LET MAX.U.HOPS = NODES.HOPPED(MSG.NR)
```

```
REGARDLESS
LET RELAY.NODE = RELAYER(MSG.NR)
DESTROY MESSAGE CALLED MSG.NR
''    GO TO THE OUTGOING UPDATE QUEUE OF THE NODE WHICH RELAYED THE ABOVE
''    UPDATE. IF THE UPDATE QUEUE IS EMPTY, DEFINE THE LINK AS IDLE
''    AND CHECK THE NODE'S PACKET QUEUE FOR PACKETS TO BE RELAYED BY
''    DELIVERED TO THIS NODE; OTHERWISE, PLACE THE NEXT UPDATE ON
''    THE LINK AND ADJUST THE UPDATE QUEUE LENGTH

FOR EACH MESSAGE IN UPDATE.OUT.QUEUE( RELAY.NODE )
WITH NEXT.STOP(MESSAGE) = THIS.NODE
FIND THE FIRST CASE
IF NONE

''    CHECK PACKET QUEUE AT THE NODE WHICH SENT THIS UPDATE FOR PACKETS
''    TO BE RELAYED BY OR DELIVERED TO THIS NODE. IF PACKET QUEUE IS
''    EMPTY, DEFINE THE LINK AS IDLE; OTHERWISE, PLACE THE
''    NEXT PACKET FOR THIS NODE ON THE LINK

FOR EACH MESSAGE IN PACKET.OUT.QUEUE( RELAY.NODE )
WITH NEXT.STOP(MESSAGE) = THIS.NODE
FIND THE FIRST CASE
IF NONE
    LET LINK.MONITOR( RELAY.NODE, THIS.NODE, 1 ) = IDLE

    IF PRNT >= 5
        SKIP 1 OUTPUT LINE
        PRINT 1 LINE WITH RELAY.NODE, THIS.NODE AND TIME.V AS FOLLOWS
LINK *** / *** IDLE AT ****.******* SEC.
    REGARDLESS

ELSE
    REMOVE MESSAGE FROM PACKET.OUT.QUEUE( RELAY.NODE )
    FILE MESSAGE IN LNK.FLAG(RELAY.NODE)
    LET LINK.ENTRY.TIME(MESSAGE) = TIME.V
    LET TOT.QUEUE.TIME(MESSAGE) = TOT.QUEUE.TIME(MESSAGE) + TIME.V
                                  - QUEUE.ENTRY.TIME(MESSAGE)
    LET QUEUE.ENTRY.TIME(MESSAGE) = 0.0

''    SCHEDULE THE ARRIVAL OF THE PACKET JUST RELEASED FROM THE QUEUE

''    SCHEDULE A PACKET.ARRIVES GIVEN MESSAGE IN PKT.XMN.TIME UNITS
    LET LINK.MONITOR( RELAY.NODE, THIS.NODE, 2 ) =
    LET LINK.MONITOR( RELAY.NODE, THIS.NODE, 2 ) = TIME.V
    LET LINK.TIMER( 1, RELAY.NODE, THIS.NODE, 1, ) = TIME.V

    IF PRNT >= 1
```

108

```
        SKIP 1 CUTPUT LINE
        PRINT 3 LINES WITH INFO1(MESSAGE), INFO2(MESSAGE), RELAY.NODE,
            THIS.NODE DESTINATICN(MESSAGE), TIME.V } RELAY.NODE AND
            LINK.MCNITOR( RELAY.NODE, THIS.NODE, 2 ) AS FOLLOWS
**** /** RELEASED FROM THE OUTGOING PACKET QUEUE OF NODE *** FOR ***
        WITH DESTINATION *** AT *****.***** SEC.
        OUTGOING PACKET QUEUE FROM NODE *** IS *******
        REGARDLESS
    ''
    ELSE
    REMOVE MESSAGE FROM UPDATE.OUT.QUEUE( RELAY.NODE )
    ::
    ::  SCHEDULE ARRIVAL CF UPDATE JUST RELEASED FROM QUEUE
    ::
        SCHEDULE AN UPDATE.ARRIVES GIVEN MESSAGE IN U.XMN.TIME UNITS
        LET LINK.MONITOR( RELAY.NODE, THIS.NODE, 4 ) =
        LET LINK.MCNITCR( RELAY.NODE, THIS.NODE, 4 ) - 1
        LET LINK.TIMER( 2, RELAY.NODE, THIS.NODE, 1 ) = TIME.V
    ''
    IF PRNT >= 4
        SKIP 1 OUTPUT LINE
        PRINT 2 LINES WITH RELAY.NODE, THIS.NODE, TIME.V, RELAY.NODE  AND
            LINK.MCNITCR( RELAY.NODE, THIS.NODE, 4 ) AS FOLLOWS
UPDATE RELEASED FROM THE OUTGOING UPDATE QUEUE OF NODE *** TO ***
        AT *****.***** SEC.
        OUTGOING UPDATE QUEUE FROM NODE *** IS *******
        REGARDLESS
    ''
    .. REGARDLESS
    ..
    RETURN
    END ''CF UPDATE.ARRIVES
    '':
    ''::   THIS ROUTINE CALCULATES CHANNEL VALUES BASED ON A TIME-WEIGHTED
    ''::   AVERAGE OF QUEUE SIZES OVER A SPECIFIED TIME CALLED THE WINDOW.
    ''::   QUEUE SIZE INFORMATICN OLDER THAN THE WINDOW TIME IS DISCARDED.
    ''
    EVENT COMPUTE.CY
    DEFINE EDGE, LAST, SUM, LAS.CU, SPAN, MIC, AREA, WEIGHT, BLOCK
        AND REMAINDER AS REAL VARIABLES
    DEFINE NONE TO MEAN 0
    ''
    IF PRNT >= 2
        SKIP 1 OUTPUT LINE
        PRINT 2 LINES WITH TIME.V AS FOLLOWS
UPDATE CYCLE BEGINS AT *****.***** SEC
```

109

```
NEW CHANNEL VALUES ARE BEING COMPUTED
    SKIP 1 OUTPUT LINE
REGARDLESS
''
FOR THIS.NODE = 1 TO N.NODE, DO
''    DESTROY QUEUE INFORMATION BEYOND WINDOW SIZE.   "PACK" IS A PACKAGE
''    OF INFORMATION DESCRIBED LATER.
''
FOR EACH PACK IN TIME.QUEUE (THIS.NODE) WITH ENTRY.TIME(PACK) <
    (TIME.V - WINDOW), DO
    REMOVE PACK FROM TIME.QUEUE (THIS.NODE)
    DESTROY PACK
LOOP
''
''    CALCULATE THE CV TO EACH NEIGHBOR
''
FOR J = 1 TO MAX.LINKS.PER.NODE, DO
  IF NEIGHBOR.LIST (THIS.NODE, J, 2) = YES
    LET NEIB = NEIGHBOR.LIST (THIS.NODE, J, 1)
    LET EDGE = C.0
    LET LAST = TIME.V
    LET SUM = 0.0
    LET LAS.QU = 0.0
    LET ANY.PACKS = NONE
    FOR EACH PACK IN TIME.QUEUE (THIS.NODE) WITH
        PAC.NEIGHBOR (PACK) = NEIB, DO
      LET ANY.PACKS = YES
      LET SPAN = LAST - ENTRY.TIME (PACK)
      LET MID = SPAN/2. + EDGE
      LET APEA = REAL.F(NUMBER(PACK)) * SPAN
      LET BLOCK = AREA
      LET SUM = SUM + BLOCK
      LET EDGE = EDGE + SPAN
      LET LAST = ENTRY.TIME(PACK)
      LET LAS.QU = REAL.F(NUMBER(PACK))
    LOOP
    IF ANY.PACKS = NONE
      LET LAS.QU = LINK.MCNITOR (THIS.NODE, NEIB, 2)
    REGARDLESS
    LET REMAINDER = WINDOW - EDGE
    LET MID = (REMAINDER/2.) + EDGE
    LET AREA = LAS.QU * REMAINDER
    LET BLOCK = AREA
    LET SUM = SUM + BLOCK
    LET CV.OF.LINK = INT.F(SUM/WINDOW + 1.)
    LET NEIGHBOR.LIST (THIS.NODE, J, 3) = CV.OF.LINK
  REGARDLESS
```

110

```
    LOOP
LOOP
''    SCHEDULE THE NEXT CV CALCULATION FOR ALL NEIGHBORS.  IN THE
''    SIMULATION, THIS PROCESS IS SYNCHRONIZED FOR EVERY NODE IN
''    THE NETWORK.
''
    SCHEDULE A ADOPT.NEW.BEST.PATH IN (2*UP.DATE.PERIOD) UNITS
''
''    EARLIEST.UPDATE AND LATEST.UPDATE SET THE NEXT INTERVAL DURING
''    WHICH ALL NODES WILL RANDOMLY INITIATE AN UPDATE MESSAGE. AFTER
''    THIS PERIOD THERE IS ANOTHER EQUAL LENGTH PERIOD WHICH ALLOWS ALL
''    UPDATE MESSAGES TO TRANSIT THE NETWORK AND REACH THEIR FINAL
''    "DESTINATIONS" BEFORE THE NEXT 'COMPUTE.CV'.
''
    LET EARLIEST.UPDATE = TIME.V + (2* UP.DATE.PERIOD)
    LET LATEST.UPDATE = TIME.V + (3* UP.DATE.PERIOD)
''
    RETURN
    END ''OF COMPUTE.CV
''
''    THIS ROUTINE ESTABLISHES THE BEST PATHS TO BE USED FOR PACKET
''       TRANSMISSION DURING THE NEXT UPDATE CYCLE.   BEST PATHS
''       ARE NOT CHANGED DURING THE UPDATE CYCLE.
''
EVENT ADOPT.NEW.BEST.PATH
FOR I = 1 TO N.NODE, DO
  FOR J = 1 TO NGFS, DO
    LET BEST.PATH ( I, J, 1 ) = BEST.PATH ( I, J, 3 )
    LET BEST.PATH ( I, J, 2 ) = BEST.PATH ( I, J, 4 )
  LOOP
LOOP
''
''    THE COMPUTE.CV EVENT IS NOW CALLED.  THIS ENSURES THAT
''    THE ORDER OF ADOPTING A NEW BEST PATH AND THEN COMPUTING
''    THE CHANNEL VALUES FOR THE NEXT UPDATE CYCLE IS PRESERVED
''    DURING THE EXECUTION OF THE SIMULATION.
''
    SCHEDULE A COMPUTE.CV NOW
''
    RETURN
    END ''OF ADOPT.NEW.BEST.PATH
''
''    THIS ROUTINE GENERATES MESSAGES MADE UP OF A RANDOM
''    NUMBER OF PKTS (BETWEEN PRESCRIBED LIMITS). PKTS ARE SENT OUT
''    ON IDLE LINKS IF AVAILABLE, OR STORED IN QUEUES IF LINKS ARE BUSY.
```

111

```
''  EVENT NEW.PACKET.MESSAGE GIVEN T.MESSAGE
DEFINE CK.XMTR, CK.RCVR, X.TOT.PERCENT AND R.TOT.PERCENT AS REAL
VARIABLES
LET X.TOT.PERCENT = 0
LET R.TOT.PERCENT = 0
LET CK.XMTR = UNIFORM.F(0.0, TRNS.PCNT, 2)
''      SELECTOR IS USED IF A PERCENTAGE OF THE MESSAGES ARE REQUIRED
''      TO BE BETWEEN NODES OF THE SAME GROUP OR FAMILY.
LET SELECTOR = UNIFORM.F(0.0, 100., 9)
''      SELECT THE TRANSMITTING NODE
FOR I = 1 TC N.NODE, DC
LET X.TOT.PERCENT = X.TOT.PERCENT + TRANSMIT.PERCENT(I)
IF CK.XMTR <= X.TOT.PERCENT
LET CK.XMTR = I
GO FIND.RECEIVER
ELSE
LOOP
''      SELECT THE RECEIVER
'FIND.RECEIVER'
LET CK.RCVR = UNIFORM.F(0.0, RCV.PCNT, 3)
FOR J = 1 TO N.NODE, DC
LET R.TOT.PERCENT = R.TOT.PERCENT + RECEIVE.PERCENT(J)
IF CK.PCVR <= R.TOT.PERCENT
LET RCVR = J
GO CK.GROUPS.ANC.FAMILIES
ELSF
LOOP
''      IF THE RECEIVER MUST BE WITHIN THE SAME GROUP OR FAMILY, KEEP
''      LOOKING UNTIL AN ACEQUATE RECEIVER IS FOUND.
'CK.GROUPS.AND.FAMILIES'
IF SELECTOR < IN.GROUP
IF GROUP(XMTR) = GRCUP(RCVR)
GO SEE.IF.XMTR.EQ.RCVR
ELSF
LET R.TOT.PERCENT = 0.0
GO FINC.RECEIVER
ELSE IF SELECTOR < (IN.GROUP + IN.FAMILY)
IF FAMILY(XMTR) = FAMILY(RCVR)
```

112

```
        GO SEE.IF.XMTR.EQ.RCVR
ELSE
        LET R.TOT.PERCENT = 0.0
        GO FIND.RECEIVER
ELSE
'SEE.IF.XMTR.EQ.RCVR'
IF RCVR = XMTR.EQ.RCVR
        LET R.TOT.PERCENT = 0.0
        GO FIND.RECEIVER
ELSE
'DERIVE.NUMBER.OF.PACKETS'
        LET PKT.COUNT = INT.F(UNIFORM.F(1.0, MAX.PKTS.PER.MSG, 4))

...     COUNT TOTAL NUMBER OF MESSAGES GENERATED. THE 'TRACER' ARRAY KEEPS
...     TRACK OF THE TOTAL NUMBER OF PACKETS GENERATED.
...
        LET NEW.MSG.TOTAL = NEW.MSG.TOTAL + 1
        LET TRACER (NEW.MSG.TOTAL, 1) = PKT.COUNT
...
...     CREATE A MESSAGE FOR EACH PACKET
...
FOR I = 1 TO PKT.COUNT, DO
    CREATE A MESSAGE
        LET TYPE(MESSAGE) = PACKET
        LET RELAYER(MESSAGE) = XMTR
        LET FG = 0
        LET GEN.COUNT = GEN.COUNT + 1
        LET ORIG.TIME(MESSAGE) = TIME.V
...
...     ADDRESS PKT TC NODE, GRCUP OR FAMILY OF DESTINATION AS APPROPRIATE
...
    IF FAMILY(XMTR) NE FAMILY(RCVR)
        LET BP.NODE = BEST.PATH (XMTR, FAMILY(RCVR), 1)
        LET FM.GP(MESSAGE) = FAMILY(RCVR)
        LET FG = FAMILY(RCVR)
        GO ADC.DESTINATICN
    ELSE
    IF GROUP(XMTR) NE GRCUP(RCVR)
        LET BP.NODE = BEST.PATH (XMTR, GROUP(RCVR), 1)
        LET FM.GP(MESSAGE) = GROUP (RCVR)
        LET FG = GRCUP(RCVR)
        GO ADC.DESTINATICN
    ELSE
        LET BP.NODE = BEST.PATH (XMTR, RCVR, 1)
'ADC.DESTINATICN'
        LET DESTINATICN(MESSAGE) = RCVR
        LET TRANS.NUMBER(MESSAGE) = NEW.MSG.TOTAL
        LET PACK.NUMBER(MESSAGE) = I
```

```
              LET NEXT.STOP(MESSAGE) = BP.NODE

''
       IF PRNT >= 1
          SKIP 1 OUTPUT LINE
          PRINT 1 LINE WITH NEW.MSG.TOTAL, I, XMTR, BP.NODE, FG, RCVR
             AND TIME.V AS FOLLOWS
****/** INITIATED FROM **** THRU *** (F/G = ***) TO *** AT ****.****** SEC
       REGARDLESS

''     IF LINK TO BP NEIGHBOR IS IDLE; SEND OUT PACKET
''
       IF LINK.MONITOR (XMTR, BP.NODE, 1) = IDLE
          FILE MESSAGE IN LNK.FLAG(XMTR)
          LET LINK.ENTRY.TIME(MESSAGE) = TIME.V
          SCHEDULE A PACKET.ARRIVES GIVEN MESSAGE IN PKT.XMN.TIME UNITS
          LET LINK.MONITOR(XMTR, BP.NODE, 1) = BUSY
          LET LINK.TIMER( , XMTR, BP.NODE, 1) = TIME.V
          LET RELEASE.TIME (MESSAGE) = TIME.V

''
       IF PRNT >= 1
          SKIP 1 OUTPUT LINE
          PRINT 2 LINES WITH LINK.MONITOR(XMTR, BP.NODE, 2) AS FOLLOWS
          FIRST PACKET OF MESSAGE DID NOT WAIT IN OUTGOING PACKET QUEUE
          OUTGOING PACKET QUEUE IS *********
       REGARDLESS

''
       IF PRNT >= 5
          SKIP 1 OUTPUT LINE
          PRINT 1 LINE WITH XMTR, BP.NODE, TRANS.NUMBER(MESSAGE),
          PACK.NUMBER(MESSAGE) AND TIME.V AS FOLLOWS
*** / *** BUSY TRANSMITTING PACKET ****/** AT ****.****** SEC
       REGARDLESS

ELSE
''     IF LINK IS BUSY, STORE PKT AT END OF THE OUTGOING PACKET QUEUE
''        FOR THAT LINK
''
          LET QUEUE.ENTRY.TIME(MESSAGE) = TIME.V
          FILE MESSAGE IN PACKET.OUT.QUEUE(XMTR)
          LET LINK.MONITOR(XMTR, BP.NODE, 2) = 1
          LET LINK.MONITOR(XMTR, BP.NODE, 2) + 1
          IF LINK.MONITOR(XMTR,BP.NODE,2) > LINK.MONITOR(XMTR,BP.NODE,3)
             LET LINK.MONITOR(XMTR,BP.NODE,3) = LINK.MONITOR(XMTR,BP.NODE,2)
          REGARDLESS

''
       IF PRNT >= 1
          SKIP 1 OUTPUT LINE
```

114

```
        PRINT 2 LINES WITH NEW.MSG.TOTAL, I, XMTR, TIME.V, XMTR AND
              LINK.MONITOR(XMTR, BP.NODE, 2 ) AS FOLLOWS
****/** ENTERS OUTGOING PACKET QUEUE AT NODE *** AS AT ****.****** SEC.
        OUTGOING PACKET QUEUE AT NODE *** IS ********
        REGARDLESS

''    IF LINK QUEUE CHANGES, CREATE A "PACK" (A PACKAGE OF INFORMATION)
''    WHICH CONTAINS THE NEW QUEUE SIZE AND THE TIME IT WAS CHANGED
''    FOR THE LATER CALCULATION OF CV

        CREATE A PACK
        LET NUMBER(PACK) = LINK.MONITOR (XMTR,BP.NODE,2)
        LET ENTRY.TIME(PACK) = TIME.V
        LET PAC.NEIGHBOR(PACK) = BP.NODE
        FILE PACK IN TIME.QUEUE(XMTR)
        REGARDLESS

''  LOOP
''    RESCHEDULE NEXT TRAFFIC SESSION UP TO MAX SET BY TRAFFIC.LIMIT

IF NEW.MSG.TOTAL <= TRAF.LIMIT
  SCHEDULE A NEW.PACKET.MESSAGE GIVEN PACKET IN
  EXPONENTIAL.F(MSG.GENERATION.INTERVAL, 1) UNITS
ELSE
  SKIP 1 OUTPUT LINE
  PRINT 2 LINES WITH TIME.V AS FOLLOWS
TRAFFIC GENERATION FOR THIS RUN = TRAFFIC LIMIT
SIMULATION TIME = ****.******
REGARDLESS

RETURN
END ''CF NEW.PACKET.MESSAGE

''    THIS ROUTINE PROCESSES PKTS AS THEY ARRIVE IN A NODE

EVENT PACKET.ARRIVES GIVEN ID.NUMBER
DEFINE MSG.NR, THIS.NODE AND PAST.NODE AS VARIABLES
DEFINE OCCURANCE AS VARIABLE
LET MSG.NR = ID.NUMBER
LET THIS.NODE = NEXT.STOP(MSG.NR)
LET PAST.NODE = RELAYER(MSG.NR)

REMOVE MSG.NR FROM LNK.FLAG(PAST.NODE)
LET LINK.ENTRY.TIME(MSG.NR) = 0.0
''    CALCULATE TOTAL TIME LINK IS BUSY
```

115

```
      LET LINK.TIMER( 1, PAST.NODE, THIS.NODE, 2) = TIME.V
      LET LINK.TIMER( 1, PAST.NODE, THIS.NODE, 3) =
      LET LINK.TIMER( 1, PAST.NODE, THIS.NODE, 3) =    +
      LET LINK.TIMER( 1, PAST.NODE, THIS.NODE, 2) =    -
      LET LINK.TIMER( 1, PAST.NODE, THIS.NODE, 1) =
''
   IF PRNT >= 1
      SKIP 1 OUTPUT LINE
      PRINT 1 LINE WITH TRANS.NUMBER(MSG.NR), PACK.NUMBER(MSG.NR),
      RELAYER(MSG.NR), NEXT.STCP(MSG.NR), TIME.V AS FOLLOWS
***/*** ARRIVES FROM *** INTO *** AT ****.******* SEC.
   REGARDLESS
''
''    IF THE PKT HAS REACHED ITS DESTINATION, GO TO A PROCESSING ROUTINE
''
   IF NEXT.STCP(MSG.NR) = DESTINATION(MSG.NR)
''
      LET TRANSIT.TIME = TRANSIT.TIME + TIME.V - ORIG.TIME(MSG.NR)
      LET DEST.COUNT = DEST.COUNT + 1
''
..    SCHEDULE A COMPLETED.TRIP GIVEN MSG.NR NEXT
''
   IF PRNT >= 1
      PRINT 1 LINE AS FOLLOWS
      AND STOPS
   REGARDLESS
''
''      IF PKT IS TO CONTINUE, ADDRESS IT TO THE NEXT BP NEIGHBOR BASED
''      ON THE NODE, GROUP OR FAMILY ID OF THE DESTINATION AS APPROPRIATE.
   ELSE
''
''      COUNT NUMBER OF PACKETS RELAYED BY THIS.NODE
''
      LET PKTS.RELAYED(THIS.NODE) = PKTS.RELAYED(THIS.NODE) + 1
      LET RELAYER(MSG.NR) = THIS.NODE
      LET FM.GP(MSG.NR) = 0
      LET FG = 0
   IF FAMILY (THIS.NODE) NE FAMILY (DESTINATION (MSG.NR))
      LET FG = FAMILY (DESTINATION (MSG.NR))
      LET BP.CBJ = FG
      LET FM.GP (MSG.NR) = FG
      GO ASGN.NEXT.STOP
   ELSE
   IF GROUP (THIS.NODE) NE GROUP (DESTINATION (MSG.NR))
      LET FG = GROUP (DESTINATION (MSG.NR))
      LET BP.CBJ = FG
```

```
          LET FM.GP (MSG.NR) = FG
     ELSE
          GO ASGN.NEXT.STOP
          LET BP.OBJ = DESTINATION (MSG.NR)
'ASGN.NEXT.STOP'
          LET NEXT.STOP(MSG.NR) = BEST.PATH (THIS.NODE, BP.OBJ,1)
          LET NODES.HOPPED(MSG.NR) = NODES.HOPPED(MSG.NR) + 1

''        SCHEDULE A PROCESSING COMPLETION TIME WHEN THE PKT WILL BE READY
''        FOR RETRANSMISSION. QUEUE.ENTRY.TIME STARTS BEFORE THE MESSAGE
''        IS PROCESSED IN ORDER TO ALLOW THE PROCESSING TIME FOR PACKETS
''        WHICH ARE NOT IN QUEUE AND NOT DELIVERED AT THE END OF THE
''        SIMULATION TO BE COUNTED AS PART OF THE TIME THE PACKET SPENT
''        IN THE NETWORK.

          LET QUEUE.ENTRY.TIME(MSG.NR) = TIME.V
          SCHEDULE A CON.PACKET.MESSAGE GIVEN MSG.NR IN PROCESSING.TIME UNITS
REGARDLESS

''        GO TO THE QUEUEING UPDATE QUEUE OF THE NODE WHICH RELAYED THE ABOVE
''        PACKET. IF THE UPDATE QUEUE IS EMPTY, CHECK THAT NODE'S
''        PACKET QUEUE TO SEE IF ADDITIONAL PACKETS ARE WAITING FOR
''        RELAY OR DELIVERY TO THIS NODE AND SCHEDULE THE ARRIVAL OF
''        THE NEXT PACKET AT THIS NODE. OTHERWISE REMOVE THE
''        WAITING UPDATE FROM THE QUEUE AND SCHEDULE ITS ARRIVAL
''        AT THIS NODE

FOR EACH MESSAGE IN UPDATE.OUT.QUEUE(PAST.NODE)
WITH NEXT.STOP(MESSAGE) = THIS.NODE
FIND THE FIRST CASE
IF NONE
          GO CHECK.PACKET.OUT.QUEUE
ELSE
          REMOVE MESSAGE FROM UPDATE.OUT.QUEUE(PAST.NODE)
          SCHEDULE AN UPDATE.ARRIVES GIVEN MESSAGE IN U.XMN.TIME UNITS
          LET LINK.MONITOR(PAST.NODE, THIS.NODE, 4) =
             LINK.MONITOR(PAST.NODE, THIS.NODE, 4 ) - 1
          LET LINK.TIMER( 2, PAST.NODE, THIS.NODE, 1 ) = TIME.V
          LET LINK.MONITOR( PAST.NODE, THIS.NODE, 1 ) = BUSY

''        IF PRNT >= 4
          SKIP 1 OUTPUT LINE
          PRINT 3 LINES WITH PAST.NODE, THIS.NODE, TIME.V, PAST.NODE AND
             LINK.MONITOR(PAST.NODE, THIS.NODE, 4 ) AS FOLLOWS
          ***    OUTGOING C.DATE QUEUE AT NODE ***  TO ***
UPDATE RELEASED FROM OUTGOING SEC.
          *******  OUTGOING UPDATE QUEUE AT NODE ***  IS *******
          REGARDLESS
```

117

```
    IF PRNT >= 5
    SKIP 1 OUTPUT LINE
    PRINT 1 LINE WITH PAST.NODE, THIS.NODE, TIME.V AS FOLLOWS
      LINK **** / *** BUSY WITH UPDATE AT ****.****** SEC.
    REGARDLESS
''
    GO NO.PACKET.XMT
''
'CHECK.PACKET.OUT.QUEUE'
'''     GO TO THE OUTGOING PACKET QUEUE OF THE NODE WHICH RELAYED
'''     THE ABOVE PACKET. IF EMPTY, DEFINE THE LINK AS IDLE; IF NOT,
'''     PLACE THE NEXT PACKET ON THE LINK AND ADJUST THE QUEUE
'''     INFORMATION BY CREATING A NEW "PACK"
''
FOR EACH MESSAGE IN PACKET.OUT.QUEUE(PAST.NODE)
WITH NEXT.STOP(MESSAGE) = THIS.NODE
FIND THE FIRST CASE
IF NONE
    LET LINK.MONITOR( PAST.NODE, THIS.NODE, 1 ) = IDLE
''
    IF PRNT >= 5
    SKIP 1 OUTPUT LINE
    PRINT 1 LINE WITH PAST.NODE, THIS.NODE AND TIME.V AS FOLLOWS
      LINK **** / *** IDLE AT ****.****** SEC.
    REGARDLESS
''
    ELSE
    REMOVE MESSAGE FROM PACKET.OUT.QUEUE(PAST.NODE)
    LET TOT.QUEUE.TIME(MESSAGE) = TOT.QUEUE.TIME(MESSAGE) + TIME.V
                                  - QUEUE.ENTRY.TIME(MESSAGE)

    LET QUEUE.ENTRY.TIME(MESSAGE) = 0.0
    FILE MESSAGE IN LINK.FLAG(RELAYER(MESSAGE))
    LET LINK.ENTRY.TIME(MESSAGE) = TIME.V
    LET LINK.MONITOR( PAST.NODE, THIS.NODE, 2) =
      LINK.MONITOR( PAST.NODE, THIS.NODE, 2) - 1
    CREATE A PACK
    LET NUMBER(PACK) = LINK.MONITOR ( PAST.NODE, THIS.NODE, 2)
    LET ENTRY.TIME(PACK) = TIME.V
    LET PAC.NEIGHBOR(PACK) = THIS.NODE
    FILE PACK IN TIME.QUEUE(PAST.NODE)
    IF NODES.HOPPED(MESSAGE) = 0      LET
      RELEASE.TIME (MESSAGE) = TIME.V
    REGARDLESS
    LET LINK.TIMER( 1, PAST.NODE, THIS.NODE, 1) = TIME.V
''
''      SCHEDULE THE ARRIVAL OF THE PKT JUST RELEASED FROM THE QUEUE.
```

118

```
''    SCHEDULE A PACKET.ARRIVES GIVEN MESSAGE IN PKT.XMN.TIME UNITS
''
      IF PRNT >= 1
        SKIP 1 OUTPUT LINE
        PRINT 4 LINES WITH TRANS.NUMBER(MESSAGE), PACK.NUMBER(MESSAGE),
          RELAYER(MESSAGE), TIME.V, NEXT.STOP(MESSAGE),
          DESTINATION(MESSAGE), FM.GP(MESSAGE), RELAYER(MESSAGE) AND
          LINK.MONITOR( PAST.NODE,THIS.NODE, 2 ) AS FOLLOWS ***
****/** RELEASED FROM OUTGOING PACKET QUEUE AT NODE ***
      AT ****.****** SEC
          PACKET ENROUTE TO NODE *** FOR DESTINATION *** ( F/G = *** )
          OUTGOING PACKET QUEUE AT NODE *** IS ********
      REGARDLESS
''
''    REGARDLESS
    'NO.PACKET.XMT'
    RETURN
    END ''OF PACKET.ARRIVES
''
''    THIS ROUTINE CONTINUES THE PACKET ON IT BEST PATH AFTER PROCESSING
''    IF THE LINK IS IDLE, OR PLACES IT IN A QUEUE.
''
EVENT CON.PACKET.MESSAGE GIVEN IDENT.MESSAGE.NUMBER
''
DEFINE MSG.NR AND THIS.NODE AS VARIABLES
LET MSG.NR = IDENT.MESSAGE.NUMBER
LET THIS.NODE = RELAYER(MSG.NR)
''
''    IF LINK IS AVAILABLE, SEND OUT PKT.   LIST LINK AS BUSY.
''
IF LINK.MONITOR ( THIS.NODE, NEXT.STOP(MSG.NR), 1) = IDLE
LET TOT.QUEUE.TIME(MSG.NR) = TOT.QUEUE.TIME(MSG.NR) + TIME.V
            - QUEUE.ENTRY.TIME(MSG.NR)
LET QUEUE.ENTRY.TIME(MSG.NR) = 0.0
FILE MSG.NR IN LNK.FLAG(THIS.NODE)
LET LINK.ENTRY.TIME(MSG.NR) = TIME.V
SCHEDULE A PACKET.ARRIVES GIVEN MSG.NR IN PKT.XMN.TIME UNITS
LET LINK.TIMER( 1, THIS.NODE, NEXT.STOP(MSG.NR), 1) = TIME.V
''
      IF PRNT >= 1
        SKIP 1 OUTPUT LINE
        PRINT 3 LINES WITH TRANS.NUMBER(MSG.NR), PACK.NUMBER(MSG.NR),
          RELAYER(MSG.NR), NEXT.STOP(MSG.NR), DESTINATION(MSG.NR),
          FM.GP(MSG.NR), TIME.V, RELAYER(MSG.NR) AND LINK.MONITOR(RELAYER
          (MSG.NR), NEXT.STOP(MSG.NR), 2) AS FOLLOWS
****/** RELAYED *** THRU *** FOR *** (F/G = ***) AT ****.****** SEC
```

```
''          PACKET DID NOT WAIT IN OUTGOING PACKET QUEUE
            OUTGOING PACKET QUEUE FROM NODE *** IS ********
    REGARDLESS
''
''  LET LINK.MONITOR( THIS.NODE, NEXT.STOP(MSG.NR), 1) = BUSY
''
    IF PRNT >= 5
    SKIP 1 OUTPUT LINE
    PRINT 1 LINE WITH RELAYER(MSG.NR), NEXT.STOP(MSG.NR),
          TRANS.NUMBER(MSG.NR), PACK.NUMBER(MSG.NR) AND
          TIME.V AS FOLLOWS
    LINK *** / *** BUSY TRANSMITTING PACKET ****/** AT ****.****** SEC
    REGARDLESS
''
    ELSE
''
    ::  IF LINK WAS BUSY, PLACE PKT IN QUEUE AND CREATE A PACK WITH NEW
    ::     QUEUE INFORMATION.
''
    FILE MSG.NR IN PACKET.OUT.QUEUE(THIS.NODE)
    LET LINK.MONITOR (THIS.NODE, NEXT.STOP(MSG.NR),2) =
        LINK.MONITOR (THIS.NODE, NEXT.STOP(MSG.NR), 2) + 1
    IF LINK.MONITOR (THIS.NODE, NEXT.STOP(MSG.NR), 2) >
        LET LINK.MONITOR (THIS.NODE, NEXT.STOP(MSG.NR), 3) =
            LINK.MONITOR (THIS.NODE, NEXT.STOP(MSG.NR), 3)
    LINK.MONITOR (THIS.NODE, NEXT.STOP(MSG.NR), 2)
    REGARDLESS
    CREATE A PACK
    LET NUMBER(PACK) = LINK.MONITOR (THIS.NODE, NEXT.STOP(MSG.NR), 2)
    LET ENTRY.TIME(PACK) = TIME.V
    LET PAC.NEIGHBOR(PACK) = NEXT.STOP(MSG.NR)
    FILE PACK IN TIME.QUEUE (THIS.NODE)
''
    IF PRNT >= 1
    SKIP 1 OUTPUT LINE
    PRINT 2 LINES WITH TRANS.NUMBER(MSG.NR), PACK.NUMBER(MSG.NR),
          THIS.NODE, NEXT.STOP(MSG.NR), TIME.V, THIS.NODE AND
          LINK.MONITOR(THIS.NODE, NEXT.STOP(MSG.NR), 2) AS FOLLOWS
    *****/** ENTERS OUTGOING PACKET QUEUE IN **** FOR **** AT ****.****** SEC.
            OUTGOING PACKET QUEUE FROM NODE *** IS ********
    REGARDLESS
''
    REGARDLESS
    RETURN
    END ''OF CON.PACKET.MESSAGE
''
::  THIS ROUTINE COLLECTS STATISTICAL DATA WHEN A PKT REACHES ITS
```

120

```
''      DESTINATION.
''
EVENT COMPLETED.TRIP GIVEN MES.NUM
DEFINE DEL.TIME AS A REAL VARIABLE
LET MESSAGE = MES.NUM
LET CNTR = NODES.HOPPED(MESSAGE) + 1
''
IF LONGEST.PATH < CNTR
LET LONGEST.PATH = CNTR
REGARDLESS
''      PRINT ALERT IF NODES HOPPED >= TOTAL NODES IN NETWORK.
''
IF SPECIFY.OUTPUT = 0
''
IF (CNTR >= N.NODE AND MSG.HLT = 0)
   PRINT 1 LINE AS FOLLOWS
PROBLEM -- MORE HOPS THAN NODES
   LET MSG.HLT = 1
REGARDLESS
''
REGARDLESS
''      INCREMENT COUNTER FOR TOTAL NODES HOPPED FOR THIS PKT AND SUM NET
''      TIME FOR GIVEN NUMBER OF HOPS.
''
LET HOP.COUNT (CNTR, PACKET) = HOP.COUNT (CNTR, PACKET) + 1
LET DEL.TIME = TIME.V - RELEASE.TIME (MESSAGE)
LET CLOCK.DATA (CNTR, 1) = CLOCK.DATA (CNTR, 1 )+ DEL.TIME
''      NOTE IF THIS TRANSIT TIME IS A NEW MAX FOR THIS NUMBER OF HOPS.
''
IF DEL.TIME > CLOCK.DATA (CNTR, 2)
   LET CLOCK.DATA (CNTR, 2) = DEL.TIME
REGARDLESS
''      INCREMENT TRACER ARRAY WHICH KEEPS THE NUMBER OF PKTS GENERATED
''      AND THE NUMBER REACHING THEIR DESTINATION.
''
LET TRACER (TRANS.NUMBER(MESSAGE), 2) = TRACER(TRANS.NUMBER(MESSAGE),2)+1
''
DESTROY MESSAGE CALLED MES.NUM
RETURN
END ''OF COMPLETED.TRIP
''
''      THIS ROUTINE IDENTIFIES A SET OF THE BUSIEST LINKS
```

```
'''    OVER THE FIRST HALF OF THE SIMULATION SO THAT
'''    THE QUEUE SIZES OF THESE LINKS CAN BE SAMPLED
'''    DURING THE SECOND HALF OF THE TEST. CRITERIA FOR
'''    SELECTION OF THE BUSIEST LINKS IS BASED ON THE
'''    NUMBER OF UPDATES (FOR 'UP.SMP.SET(*,*)') AND
'''    NUMBER OF PACKETS (FOR 'PKT.SMP.SET(*,*)') PASSED
'''    OVER THE LINK .

   EVENT QU.SAMPLER

   IF SPECIFY.OUTPUT = 0

   SKIP 1 OUTPUT LINE
   PRINT 1 LINE AS FOLLOWS
'''----------------------------MID POINT----------------------------
   SKIP 1 OUTPUT LINE

   REGARDLESS

'''    TO THE LARGEST OUTGOING PACKET AND UPDATE QUEUE SIZES
'''    DURING THE FIRST HALF OF THE SIMULATION .

   FOR F = 1 TO N.NODE DO
   FOR T = 1 TO N.NODE, DO
      IF P.Q.MAX < LINK.MONITOR (F,T,3)
         LET P.Q.MAX = LINK.MONITOR (F,T,3)
      REGARDLESS
      IF U.Q.MAX < LINK.MONITOR (F,T,5)
         LET U.Q.MAX = LINK.MONITOR (F,T,5)
      REGARDLESS
   LOOP

'''    SMP.LINKS IS AN INPUT VARIABLE LISTING THE NUMBER OF LINKS TO BE
'''    SAMPLED FOR QUEUE SIZE .

   LET I = 1

'FILL.PKT.SMP.SET'
   FOR F = 1 TO N.NODE DO
   FOR T = 1 TO N.NODE DO
      IF LINK.MONITOR (F,T,3) = P.Q.MAX
         LET PKT.SMP.SET(I,3) = F
         LET PKT.SMP.SET(I,1) = F
         LET PKT.SMP.SET(I,2) = T
         LET I = SMP.LINKS
         GO NOW.FILL.UP.SMP.SET
      ELSE
         LET I = I+1
```

```
        ELSE GC HCP.NEW.P.Q.MAX
        IF NEW.P.Q.MAX < LINK.MCNITOR(F,T,3) AND LINK.MONITOR(F,T,3)
            < P.Q.MAX
            LET NEW.P.Q.MAX = LINK.MONITOR(F,T,3)
        REGARDLESS
'HOP.NEW.P.Q.MAX'
    LOOP
    LET P.Q.MAX = NEW.P.Q.MAX
    GO FILL.PKT.SMP.SET

'NOW.FILL.UP.SMP.SET'
    LET I = 1
'FILL.UP.SMP.SET'
    FOR F = 1 TO N-NODE, DC
    FOR T = 1 TO N-NNCDE, DC
        IF LINK.MCNITCR(F,T,5) = U.Q.MAX
            LET UP.SMP.SET(I,1) = F
            LET UP.SMP.SET(I,2) = T
            IF I = SMP.LINKS
            GO BEGIN.SAMPLING
        ELSE
            LET I = I + 1
        GO HCP.NEW.U.Q.MAX
    ELSE
        IF NEW.U.Q.MAX < LINK.MONITOR(F,T,5) AND LINK.MONITOR(F,T,5)
            < U.Q.MAX
            LET NEW.U.Q.MAX = LINK.MONITOR(F,T,5)
        REGARDLESS
'HOP.NEW.U.Q.MAX'
    LOCP
    LET U.Q.MAX = NEW.U.C.MAX
    GO FILL.UP.SMP.SET

''      SCHEDULE FIRST SAMPLE OF ALL LINKS IN SMP.SET
'BEGIN.SAMPLING'
    LET TI.MFR = TIME.LIMIT/(2.* REAL.F(NO.OF.SAMPLES))
    SCHEDULE A SAMPLE IN (EXPONENTIAL.F(TI.MER, 8)) UNITS
    RETURN
    END '' OF QU.SAMPLER

''      THIS ROUTINE SAMPLES THE LINKS IDENTIFIED IN QU.SAMPLER WITH AN
''      EXPONENTIAL SAMPLING RATE.
```

123

```
''
EVENT SAMPLE
''     COUNT ACTUAL SAMPLES TAKEN.
''
LET SMP.CNTR = SMP.CNTR + 1
''     INCREMENT COUNTER BASED ON QUEUE SIZE.
''
FOR I = 1 TO SMP.LINKS, DO
LET PKT.QU.DISTR( LINK.MONITOR( PKT.SMP.SET(I,1),
       PKT.SMP.SET(I,2), 2 ) + 1 ) = PKT.QU.DISTR( LINK.MONITOR(
       (PKT.SMP.SET(I,1), PKT.SMP.SET(I,2), 2) + 1 ) + 1
''
   LET UP.QU.DISTR( LINK.MONITOR( UP.SMP.SET(I,1),
       UP.SMP.SET(I,2), 4 ) + 1 ) = UP.QU.DISTR( LINK.MONITOR(
       (UP.SMP.SET(I,1), UF.SMP.SET(I,2), 4) + 1 ) + 1
LOOP
''     SCHEDULE THE NEXT SAMPLE.
''     TI.MER IS DEFINED IN QU.SAMPLER
''
SCHEDULE A SAMPLE IN (EXPONENTIAL.F(TI.MER,8)) UNITS
RETURN
END '' OF SAMPLE
//---- INSER /* -----
//*GO.SIMU06  CD  VOL=SER=MVS004,UNIT=3350,DISP=SHR,DSN=S1611.JEN,
//GO.SIMU08 CD  DISP=SHR,DSN=MSS.S1611.WZSN57
//*DCB=(RECFM=FB,LRECL=133,BLKSIZE=4123)
//GC.SYSIN CC *
```

124

LISTING OF ARRAYS USED IN THE SIMULATION PROGRAMS

Arrays common to all programs:

1 - dimensional arrays

     FAM.OP.GRP (no.of nodes + groups + 25)
        1st -- program ID of the i-th group

     HISTOGRAM (10)
        1st -- no. of links with utilization
            factor = i/10

     PKT.QU.DISTR (1000)
        1st -- number of times packet queue size = i

     UP.QU.DISTR (1000)
        1st -- number of times update queue size = i

    2 - dimensional arrays

     CLOCK.DATA (10 * no.of nodes, 2)
        1st -- number of hops = N
        2nd -- 1 = net time for all packets hopping
            N nodes
            2 = highest individual trip time for
             a N hop packet

```
HOP.COUNT (10 * no.of nodes, 2)
     1st -- number of hops = N
     2nd -- 1 = not used
            2 = no.of packets hopping N nodes




LINK.ABLE (no.of links, 2)
     1st -- link number
     2nd -- 1 = first end node
            2 = second end node




PKT.SMP.SET (selectedlinks to be sampled, 2)
     1st -- sample link ID number
     2nd -- 1 = 'from' node ID
            2 = 'to' node ID




TRACER (estimate of the maximum number of
           messages, 2)
     1st -- message number
     2nd -- 1 = number of packets in the message
            2 = number of packets of the message
                   which reach their destination




UP.SMP.SET (selected links to be sampled, 2)
     1st -- sample link ID number
     2nd -- 1 = 'from' node ID
            2 = 'to' node ID
```

3 - dimensional arrays

BEST.PATH (no.of nodes + groups, no.of nodes
             + groups, 4)
    1st -- 'from' node, group or family ID
    2nd -- 'to' node, group or family ID
    3rd -- 1 = current best path neighbor
           2 = channel value through the current
                best path neighbor
           3 = best path neighbor to be adopted
                during next update cycle
           4 = channel value to best path
                neighbor which will be adopted
                during next update cycle

LINK.MONITOR (2 * no.of links, 2 * no.of links, 6)
    1st -- 'from' node ID
    2nd -- 'to' node ID
    3rd -- 1 = link status (idle/busy)
           2 = current packet queue of link
           3 = max packet queue thus far
           4 = current update queue of link
           5 = max update queue thus far
           6 = no. of packets in queue at the
                simulation half

NEIGHBOR.LIST (no. of nodes, max links per node, 3)
    1st -- i-th node ID
    2nd -- ID number of link
    3rd -- 1 = neighbor node ID
           2 = link status (idle/busy)
           3 = channel value from node to neighbor

4 - dimensional array

LINK.TIMER (2, 2 * no. of links, 2 * no. of links, 3)

    1st -- 1 = packet
           2 = update
    2nd -- sending node ID
    3rd -- receiving node ID
    4th -- 1 = start time for busy link
           2 = stop time for busy link
           3 = cummulative time link is busy
           4 = cummulative time link busy up to
                 simulation half

Additional arrays for Djiksta algorithm:

2 - dimensional arrays

ADDRESS.LIST (no. of nodes, no. of nodes)
    1st -- sending node ID
    2nd -- destination node ID

ADD.RESS.LIST (no. of nodes, no. of nodes)
    1st -- sending node ID
    2nd -- destination node ID

DJIKSTRA.MATRIX (no.of nodes, no. of nodes)
    1st -- sending node ID
    2nd -- destination node ID

PATH.AVAIL (no. of nodes, no.of nodes)
    1st -- sending node ID
    2nd -- destination node ID

128

APPENDIX E

SAMPLE INPUT AND OUTPUT DATA

```
Input data set:
    1                       - node.factor
    5                       - n.node
    1.   1.   1   1         - transmit.percent( )
    1.   1.   1   1           - receive.percent( )
    1.   1.   1   1             - group( )
    1.   1.   1   1               - family( )
    1.   1.   1   1
    0.1                     - up.date.period
    0.0001                  - processing.time
    0.05                    - pkt.xmn.time
    30.                     - time.limit
    0.5                     - window ( for channel value )
    00.   00.               - in.group        - in.family
    0                       - prnt
    5     500               - smp.links       - no.of.samples
    0                       - links
    4                       - max.links.per.node
    1     2                 - node no. - node no. (defines link ends)
    1     3
    1     4
    1     5
    2     3
    2     4
    2     5
    3     4
    3     5
    4     5
```

Output data set:

################# NEW SIMULATICN ####################

MAX.FKTS.PER.MSG = 10.C0
PKTS.FER.SEC.AVE = 15C.00

LINK	-- UTILIZATION FACTOR --			PKTS IN QUEUE AT HALF	PKTS IN QUEUE AT END	WAIT TIME FOR PKTS
	FACKET	UFDATE	TOTAL			
1/ 2	.616658	.C44519	.661177	C	2	1.791269
2/ 1	.566655	.046663	.613318	10	0	1.437093
1/ 3	.6C6655	.C476C1	.654255	6	13	1.724311
3/ 1	.543231	.C45593	.588924	1	14	1.295773
1/ 4	.605999	.045454	.655453	4	0	1.741389
4/ 1	.565994	.045723	.615717	0	0	1.455954
1/ 5	.549987	.C45596	.595583	0	0	1.333076
5/ 1	.645981	.046126	.6961C7	C	0	2.107419
3/ 3	.6C6654	.C45332	.651986	11	0	1.714362
3/ 2	.623334	.C47988	.671322	0	3	1.865066
2/ 4	.656654	.C45329	.701983	4	14	2.172235
4/ 2	.579959C	.C47192	.627182	3	10	1.526905
2/ 5	.543325	.C46662	.589987	12	0	1.298548
5/ 2	.593328	.C47728	.641056	0	0	1.623242
3/ 4	.579999	.C47586	.627585	0	16	1.528421
4/ 3	.639987	.C49457	.689444	0	7	2.027481
3/ 5	.546652	.C47190	.593843	14	5	1.318638
5/ 3	.66664€	.C5C8C6	.717454	6	0	2.323752
4/ 5	.579988	.C47987	.627975	15	5	1.529767
5/ 4	.639984	.C47727	.687710	12	4	2.017254

```
AVE.P.FLCW     =    .598490
AVE.U.FLCW =      .C46913
AVE.NETWCRK.FLCW =    .645403
GEN.CCUNT =         4421
DEST.COUNT =        4314
TRANSIT.TIME =      3051.942139
STRANDED.WAIT =     49.1CC677
T.IN.LINK =         .4C1688
Q.IN.LINK =       1C.718658
CHECK.DELAY =        .7C3950
AVE.DELAY.FER.PKT =      .225555
RATIO.STRANCEC =       .024803
PKTS IN CUEUE AT FALF =           98
PKTS IN CUEUE AT ENC =            93
```

INTERVAL	NUMBER CF LINKS
0. - .1C	0
.1C - .2C	0
.2C - .3C	0
.3C - .4C	0
.4C - .5C	0
.5C - .6C	4
.6C - .7C	14
.7C - .8C	2
.8C - .9C	0
.9C - 1.CC	0

SIMULATICN STCPS

130

APPENDIX F

GRAPHICAL RESULTS

DELAY : HERITSCH ALGORITHM ; SYNCHRONOUS
NODES - 5, LINKS - 10
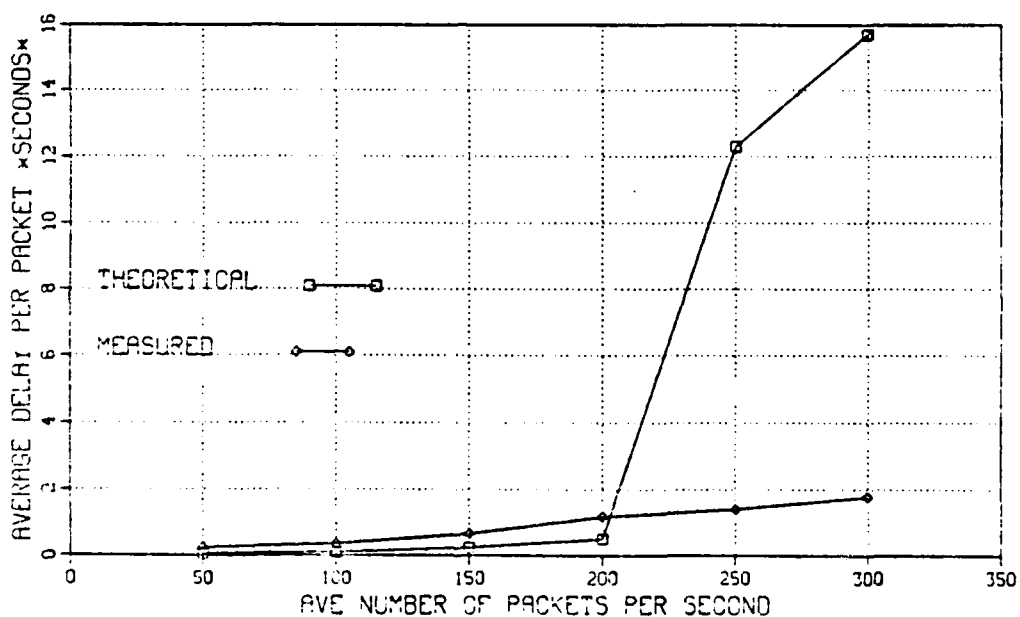
Fig. F.1.   Delay: Heritsch (synch), Network 5/10

DELAY : HERITSCH ALGORITHM ; SYNCHRONOUS
NODES - 10, LINKS - 20
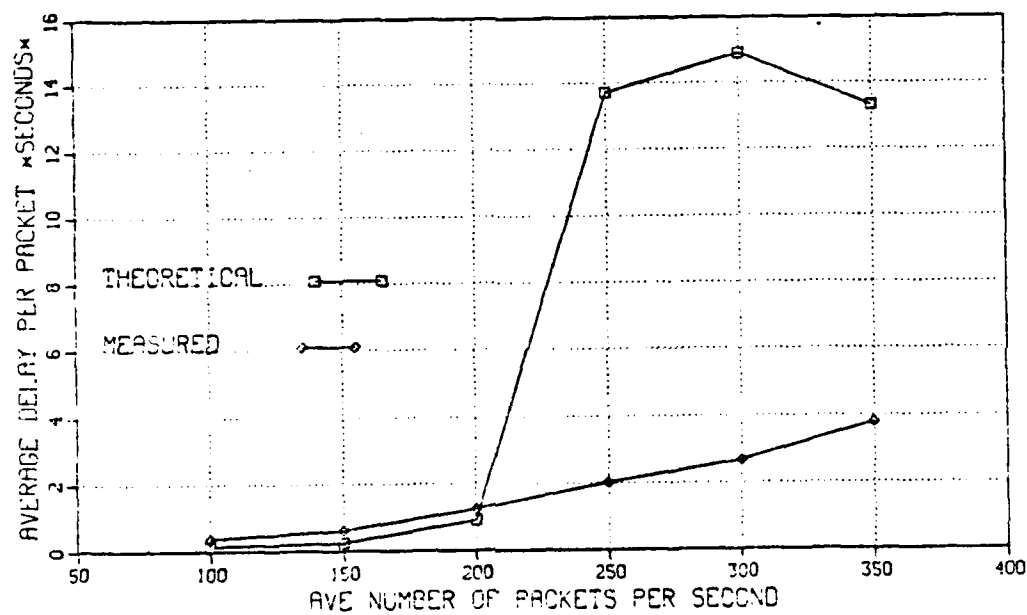
Fig. F.2. Delay: Heritsch (synch), Network 10/20

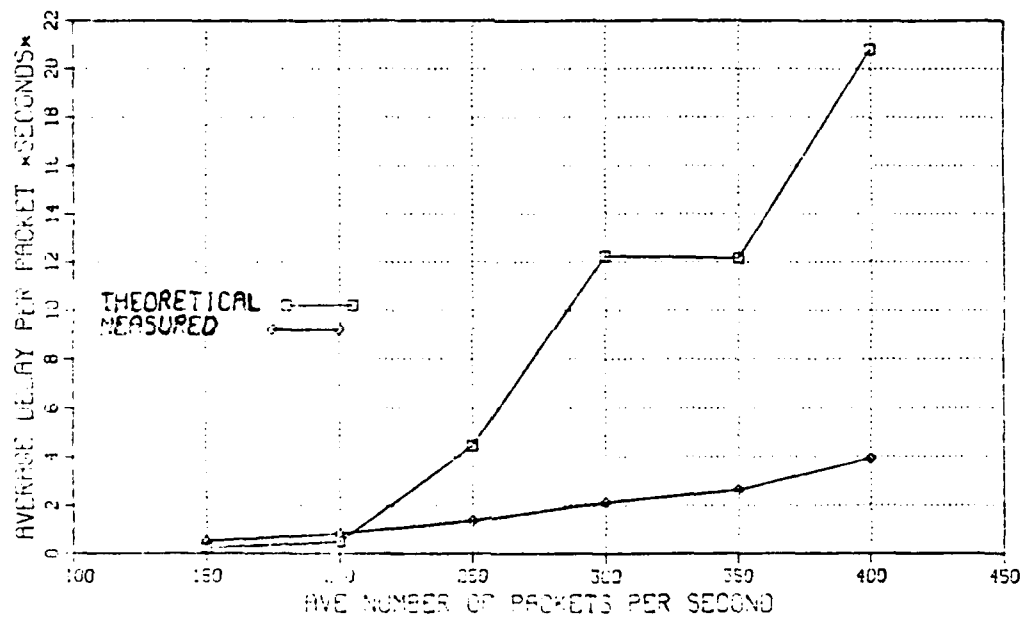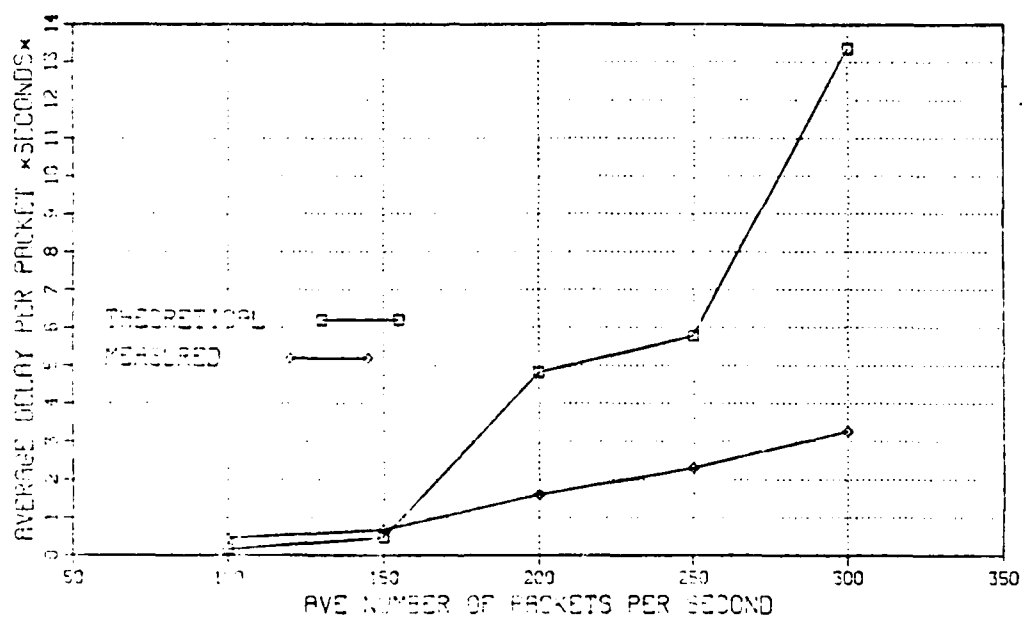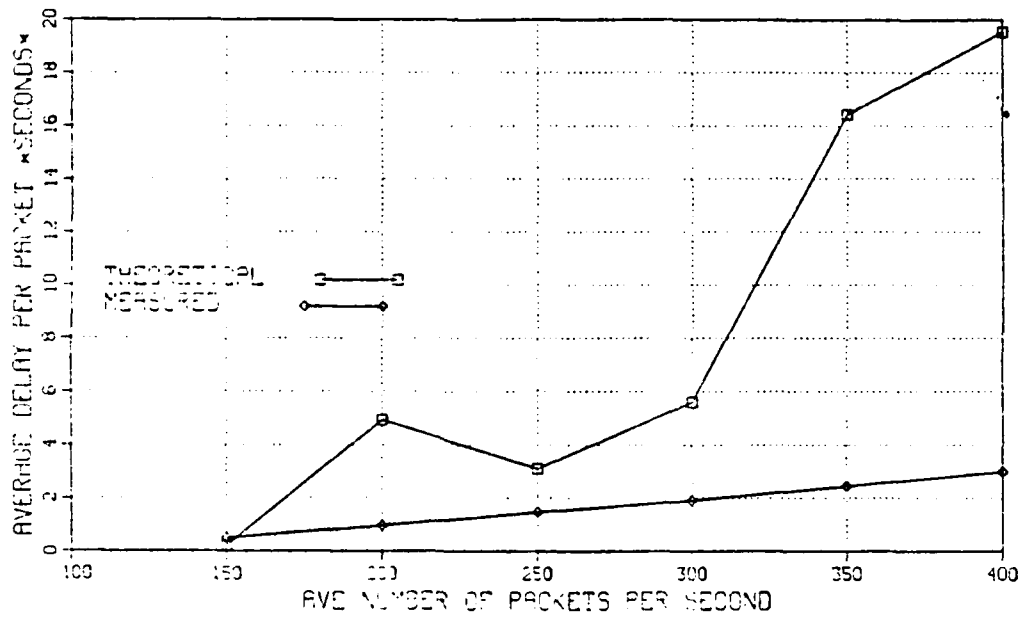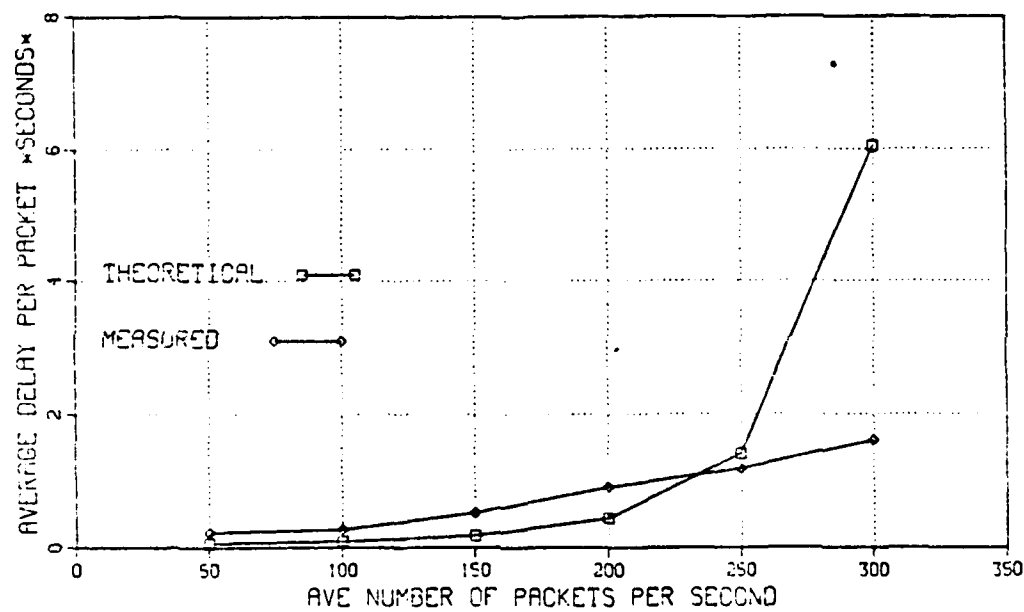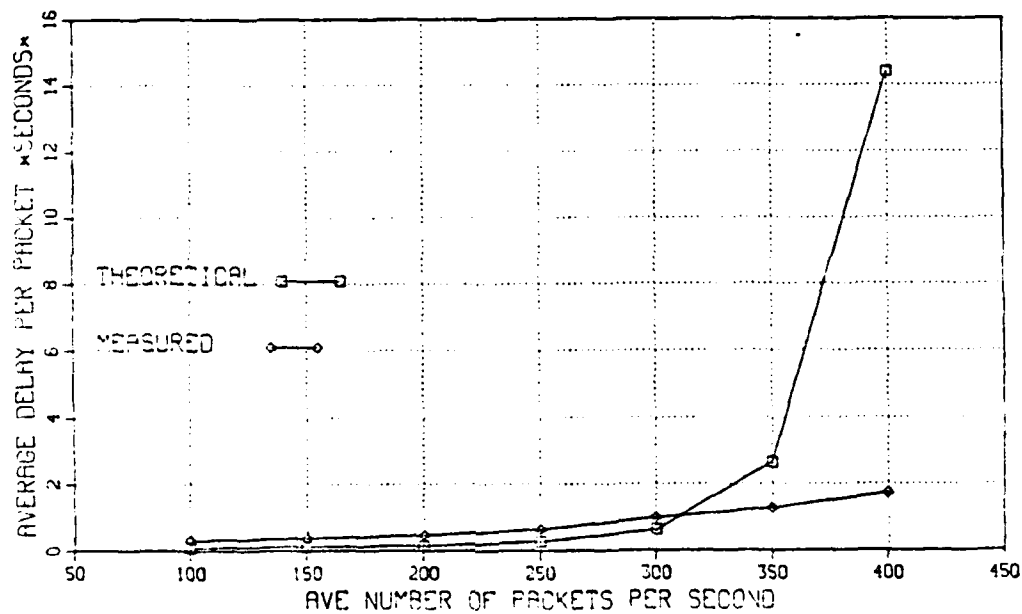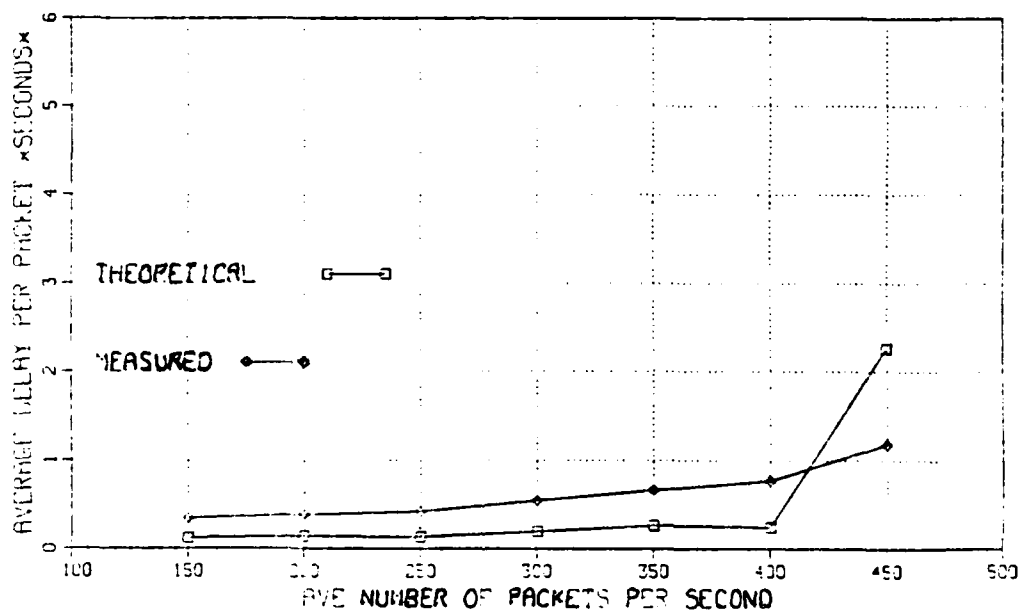DELAY : HERITSCH ALGORITHM ; SYNCHRONOUS
NODES - 15, LINKS - 30
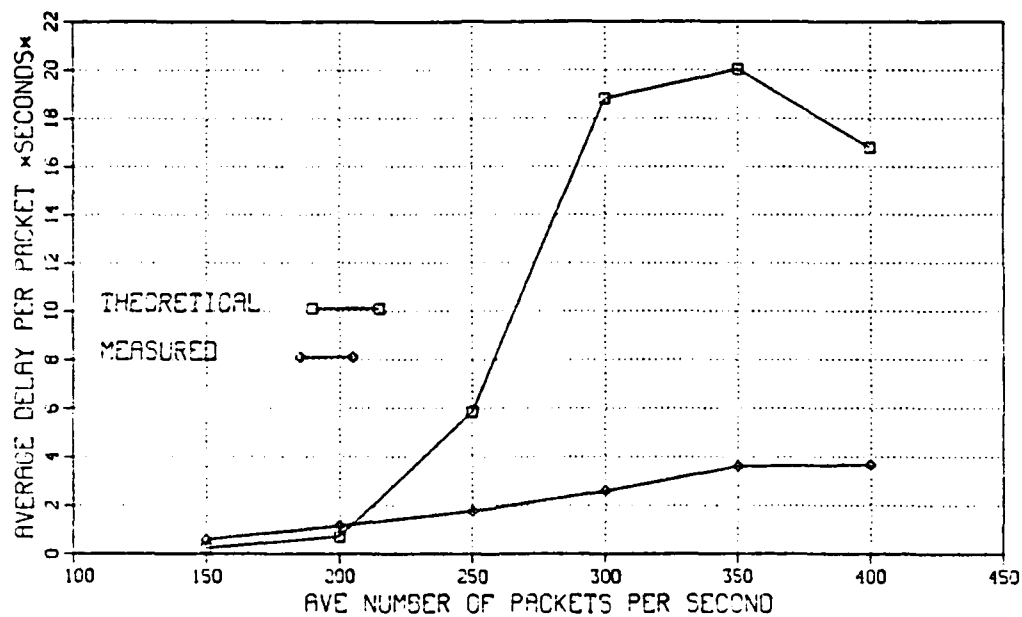
Fig. F.3. Delay: Heritsch (synch), Network 15/30

133

DELAY : HERITSCH ALGORITHM ; SYNCHRONOUS
NODES - 20, LINKS - 40

Fig. 7.4.   Delay: Heritsch (synch), Network 20/40

134
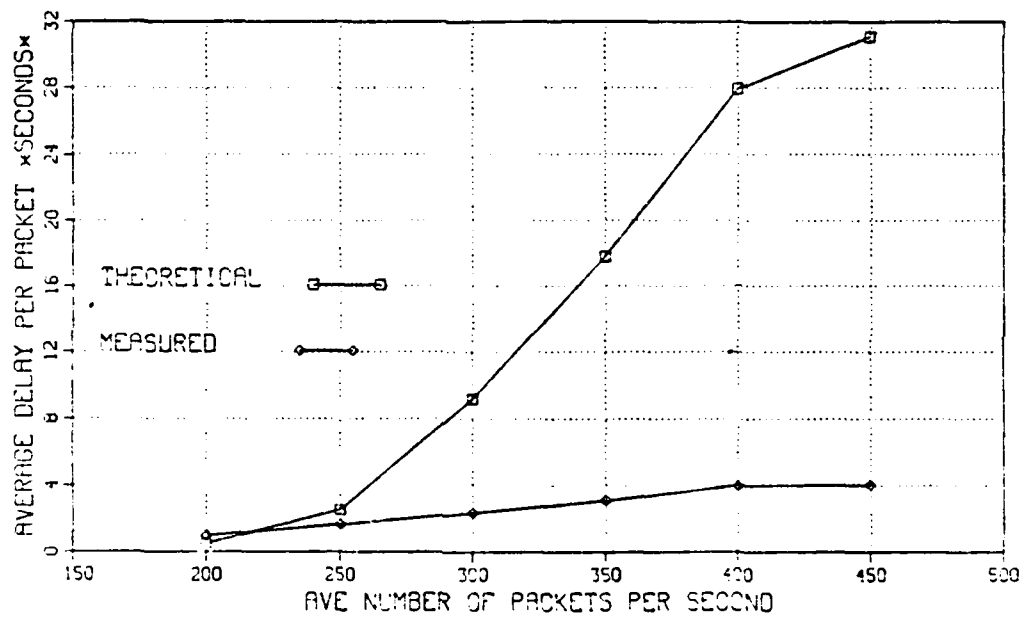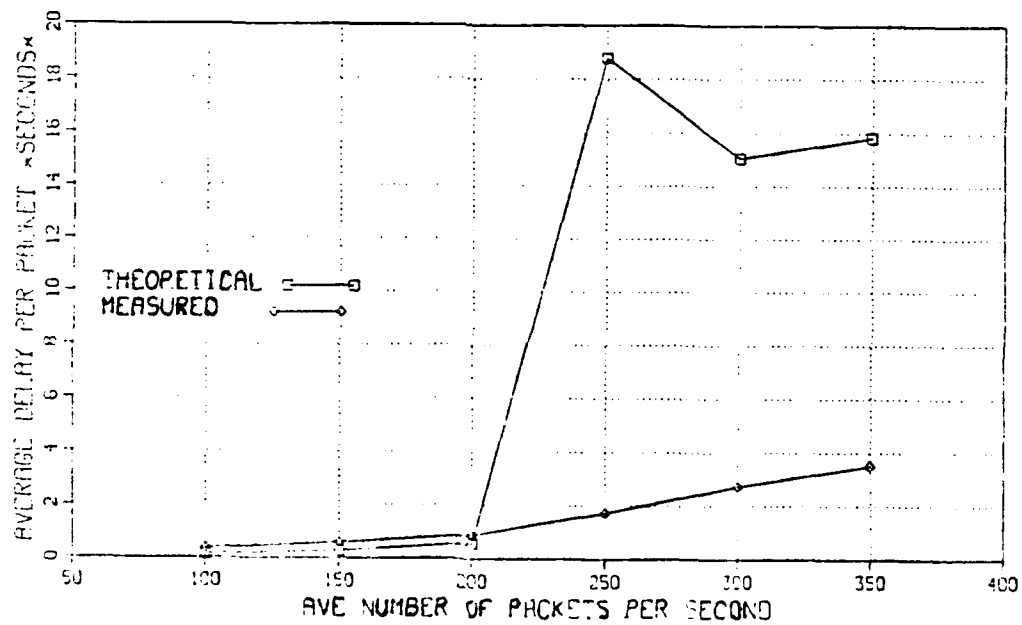
Fig. F.5.   Delay: Heritsch (synch), Network 10/20 (2 groups)

DELAY : HERITSCH ALGORITHM ; PSYNCHRONOUS
NODES - 15, LINKS - 30

Fig. F.6.   Delay: Heritsch (synch), Network 15/30 (3 groups)

DELAY : HERITSCH ALGORITHM ; SYNCH : GROUP
NODES - 10, LINKS - 20

Fig. P.7.   Delay: Heritsch (asynch), Network 10/20

DELAY : HERITSCH ALGORITHM ; SYNCH : GROUP
NODES - 15, LINKS - 30

Fig. 7.8.   Delay: Heritsch (asynch), Network 15/30

138

DELAY : DJIKSTRA ALGORITHM ; SYNCHRONOUS
NODES = 5, LINKS = 10

Fig. F.9. Delay: Djikstra, Network 5/10

DELAY : DJIKSTRA ALGORITHM ; SYNCHRONOUS
NODES - 10, LINKS - 20

Fig. F.10.   Delay: Djikstra, Network 10/20

DELAY : DJIKSTRA ALGORITHM ; SYNCHRONOUS
NODES = 15, LINKS = 30

Fig. 7.11. Delay: Djikstra, Network 15/30

DELAY : DJIKSTRA ALGORITHM ; SYNCHRONOUS
NODES - 20, LINKS - 40
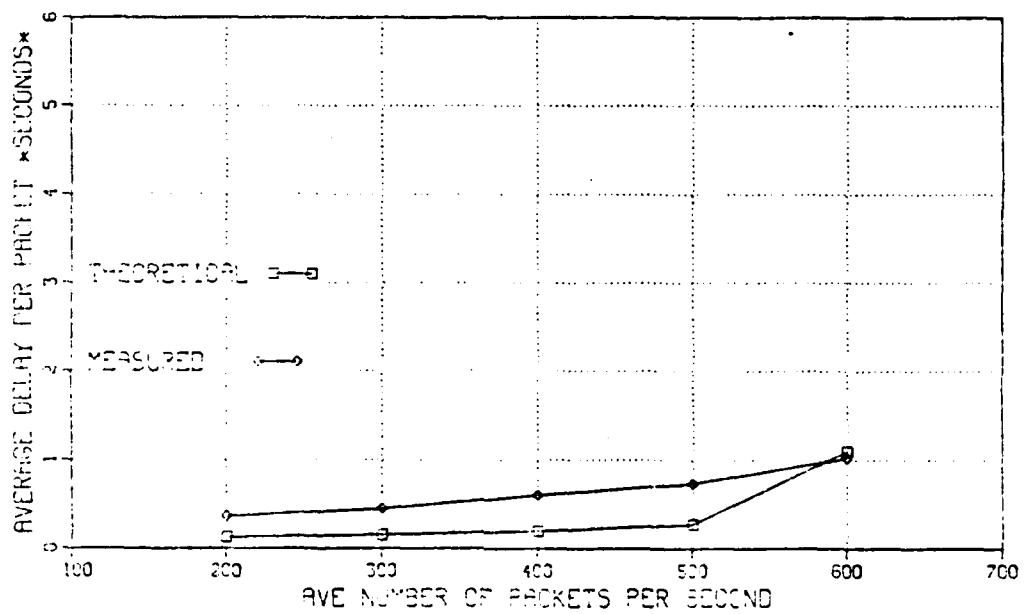
Fig. P.12.   Delay: Djikstra, Network 20/40

UTILIZATION FACTOR : HERITSCH ALGORITHM ; SYNCHRONOUS
NODES - 5, LINKS - 10
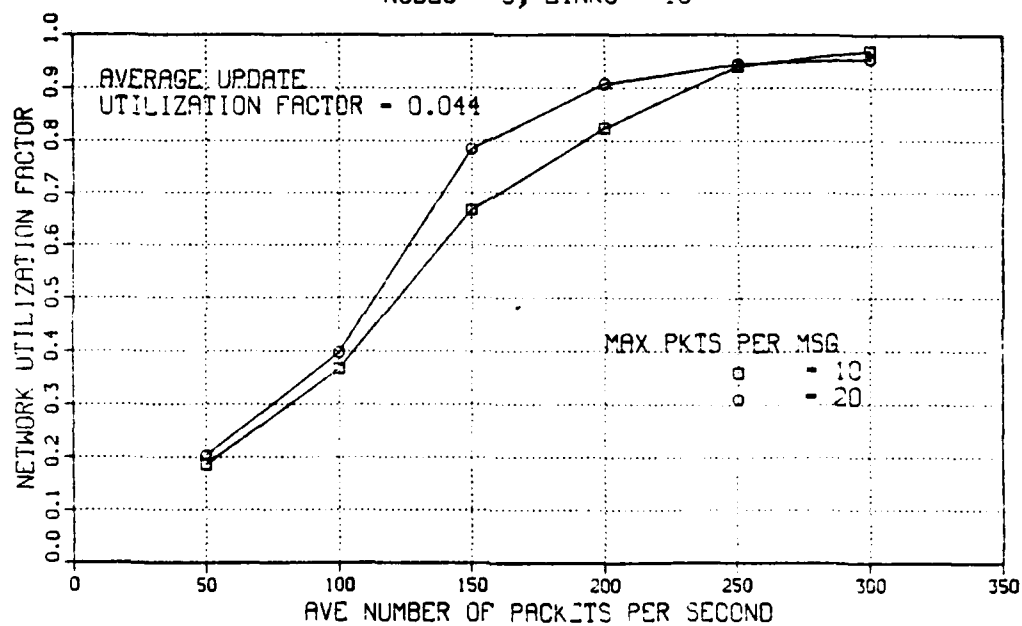
Figure F.13.   Utilization Factor: Heritsch (synch), Network
5/10

143

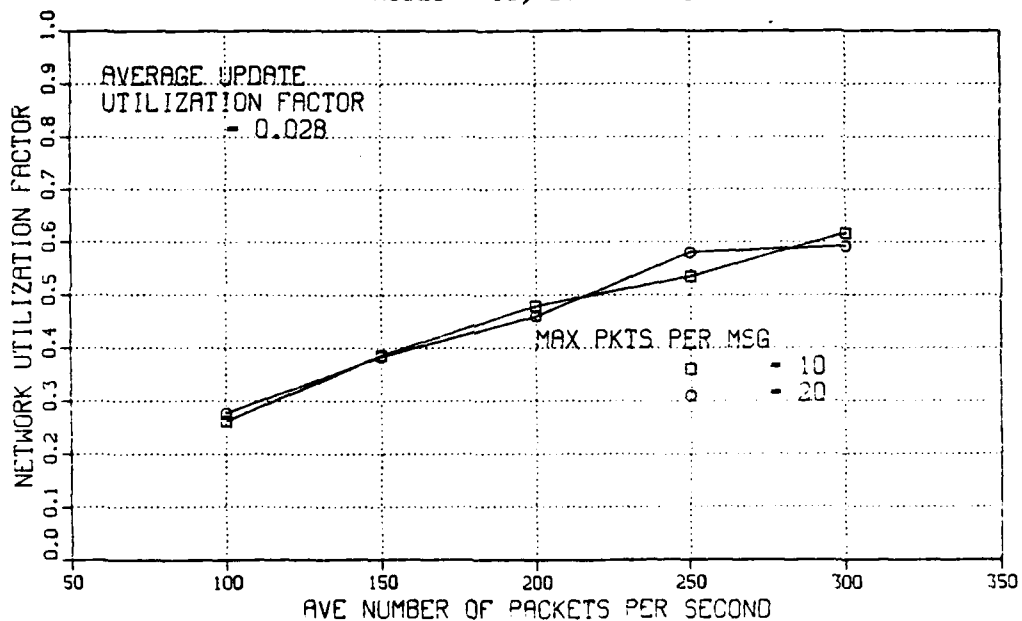UTILIZATION FACTOR : HERITSCH ALGORITHM ; SYNCHRONOUS
NODES = 10, LINKS = 20
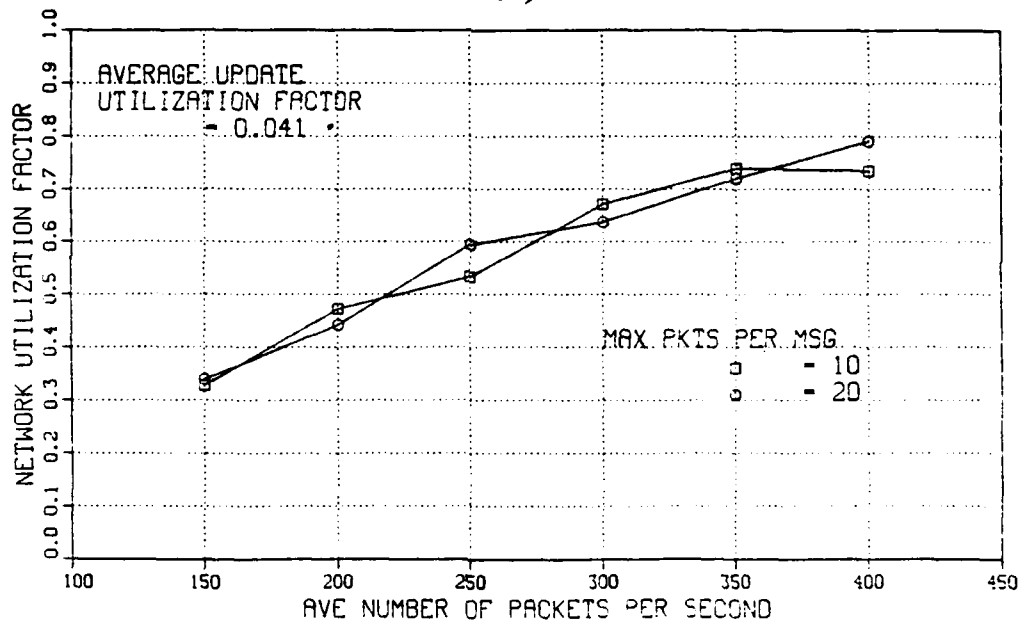
Figure F.14.   Utilization Factor: Heritsch (synch), Network
10/20

UTILIZATION FACTOR : HERITSCH ALGORITHM ; SYNCHRONOUS
NODES - 15, LINKS - 30

Figure F.15.   Utilization Factor: Heritsch (synch), Network
              15/30

145
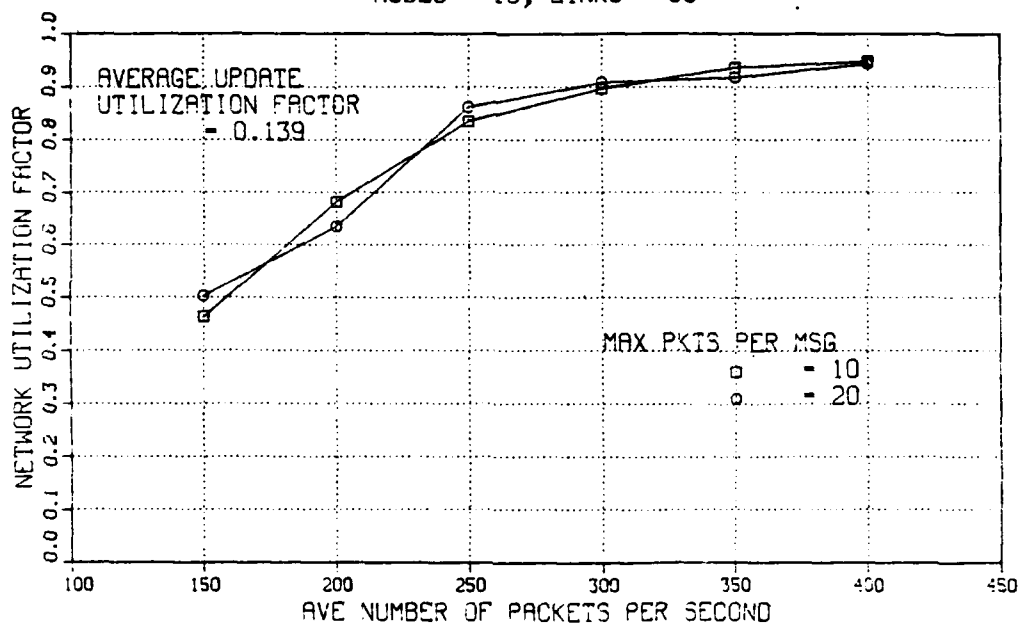
UTILIZATION FACTOR : HERITSCH ALGORITHM ; SYNCHRONOUS
NODES - 20, LINKS - 40
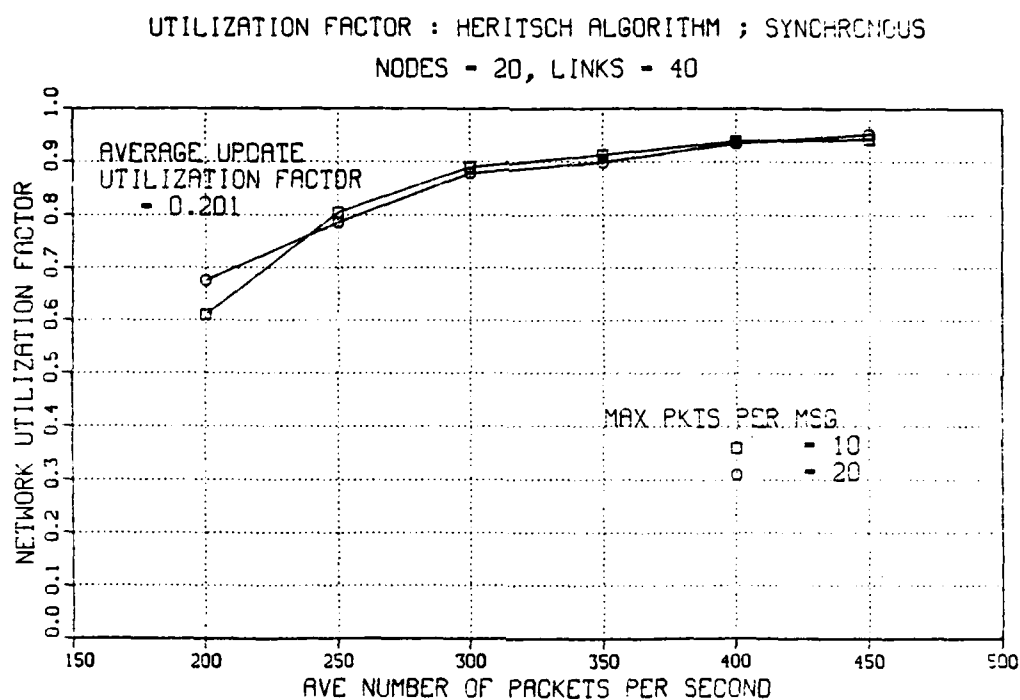
Figure F.16.   Utilization Factor: Heritsch (synch), Network
               20/40

146

UTILIZATION FACTOR : HERITSCH ALGORITHM ; SYNCH : GROUPS
NODES - 10, LINKS - 20

Figure F.17.   Utilization Factor: Heritsch (synch), Network
              10/20 (2 groups)

147

UTILIZATION FACTOR : HERITSCH ALGORITHM ; SYNCH : GROUPS
NODES - 15, LINKS - 30

Figure F.F8.   Utilization Factor: Heritsch (synch), Network
15/30 (3 groups)

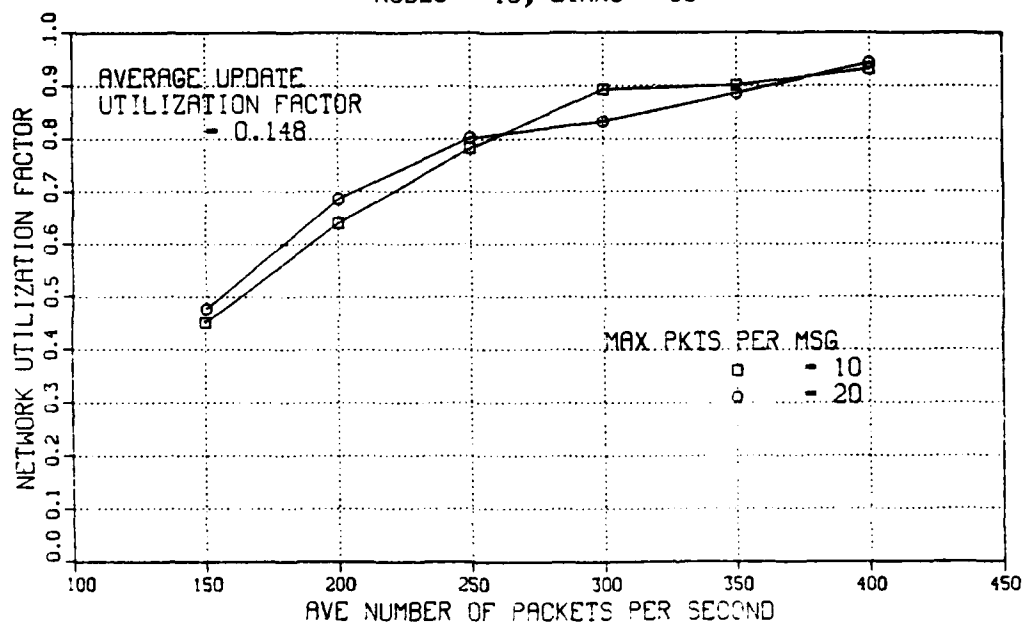UTILIZATION FACTOR : HERITSCH ALGORITHM ; ASYNCHRONOUS
NODES = 15, LINKS = 30

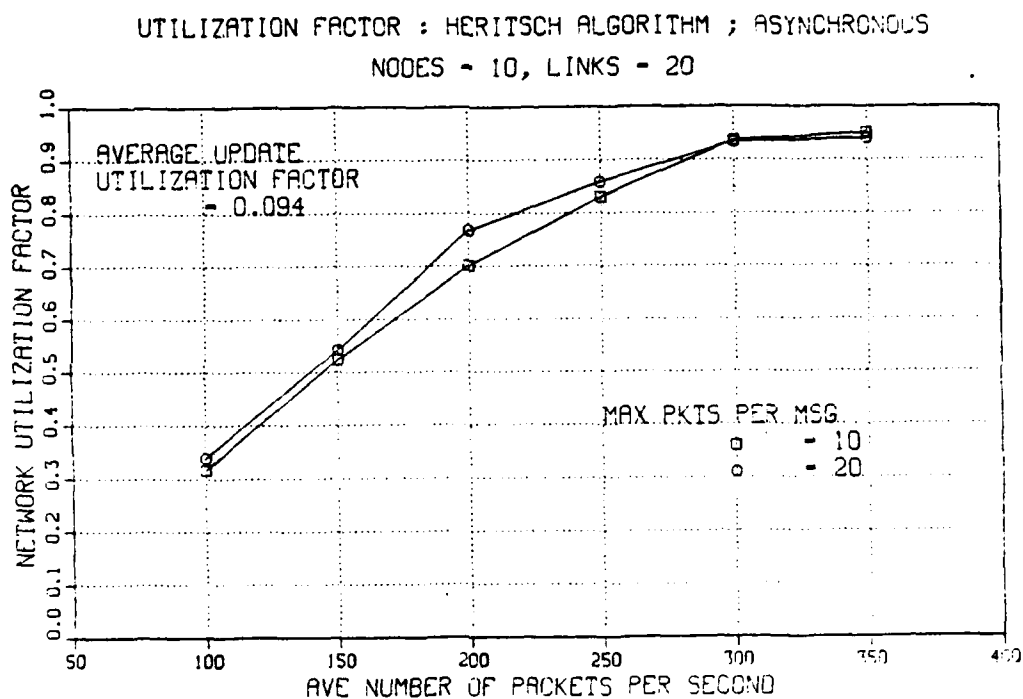Figure P.19.   Utilization Factor: Heritsch (asynch), Network
               10/20

UTILIZATION FACTOR : HERITSCH ALGORITHM ; ASYNCHRONOUS
NODES - 10, LINKS - 20

Figure F.20.   Utilization Factor: Heritsch (asynch), Network
15/30

UTILIZATION FACTOR : DJIKSTRA ALGORITHM ; SYNCHRONOUS
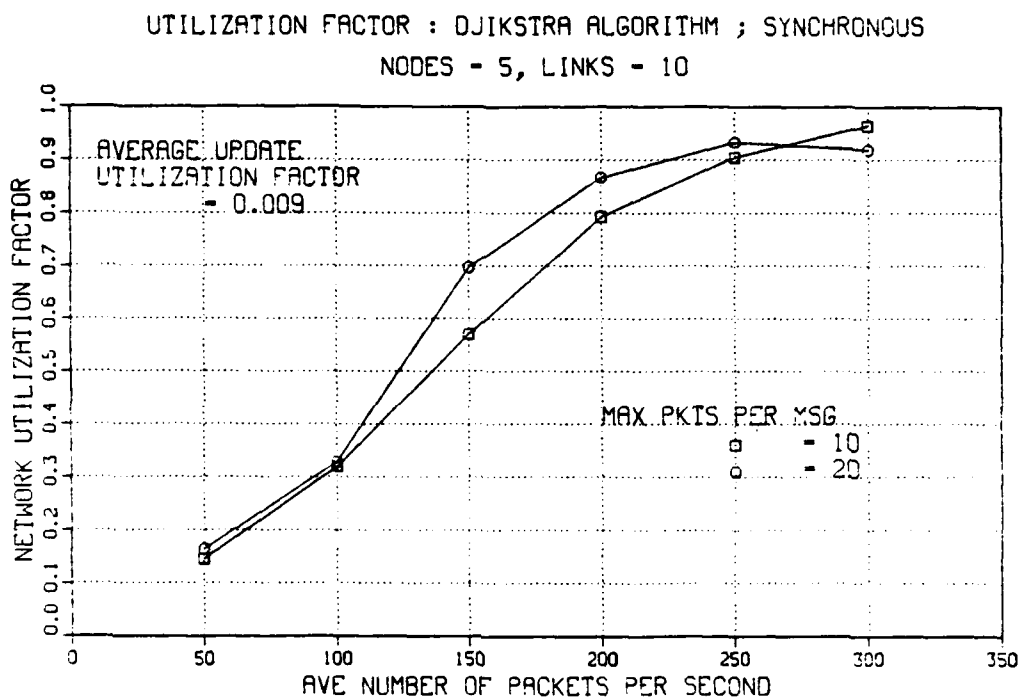NODES - 5, LINKS - 10

Fig. F.21. Utilization Factor: Djikstra, Network 5/10

UTILIZATION FACTOR : DJIKSTRA ALGORITHM ; SYNCHRONOUS
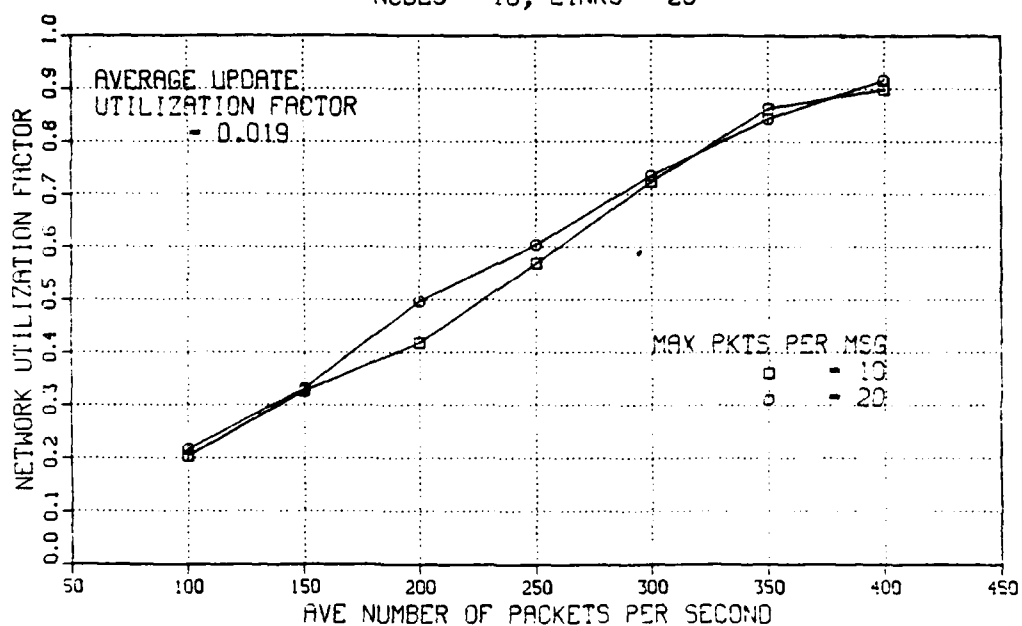• NODES - 10, LINKS - 20

Fig. F.22. Utilization Factor: Djikstra, Network 10/20

UTILIZATION FACTOR : DJIKSTRA ALGORITHM ; SYNCHRONOUS
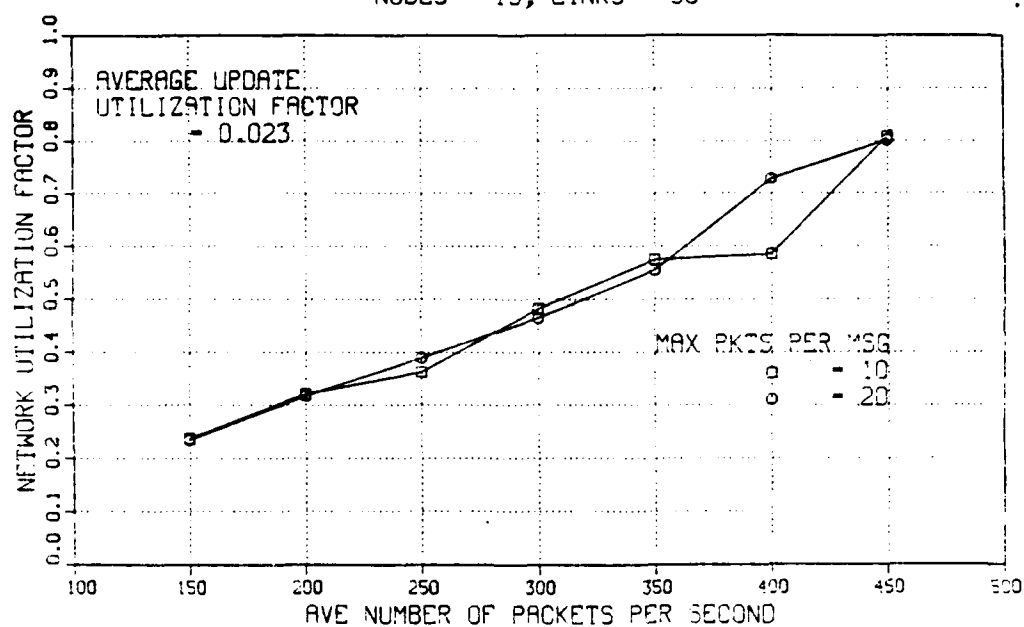NODES - 15, LINKS - 30

Fig. F.23. Utilization Factor: Djikstra, Network 15/30

UTILIZATION FACTOR : DJIKSTRA ALGORITHM ; SYNCHRONOUS
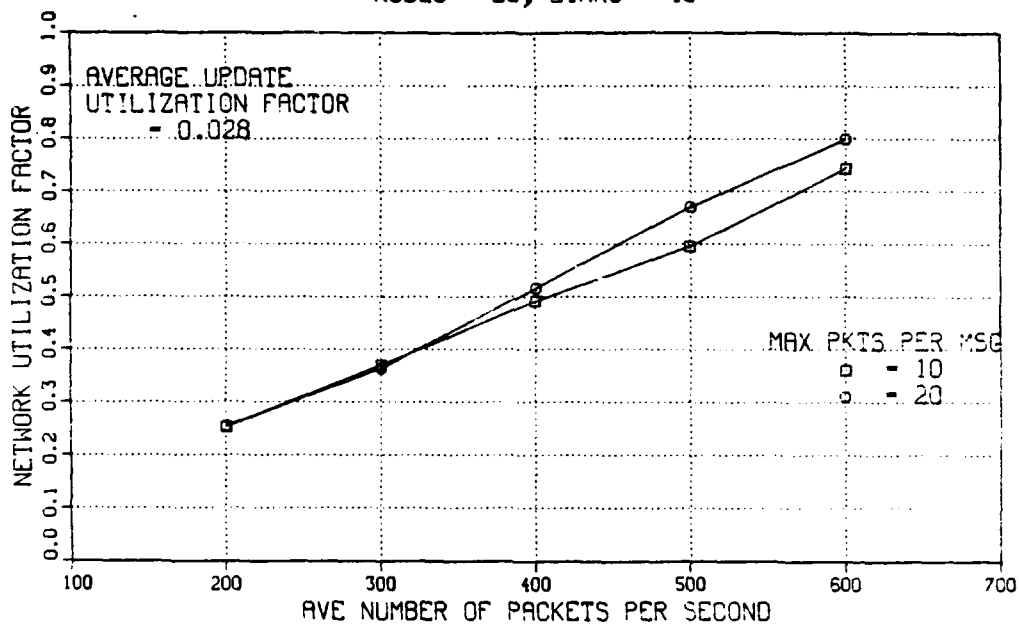
NODES - 20, LINKS - 40

Fig. F.24.   Utilization Factor: Djikstra, Network 20/40

LIST OF REFERENCES

1. Tanenbaum, Andrew, S., _Computer Networks_, p.16-21, Prentice Hall Inc., 1981.

2. Heritsch, Robert, _A Distributed Routing Protocol for a Packet Radio Network_, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1982.

3. Djikstra, E.W., "A Note on Two Problems in Connexion with Graphs" _Numerische Mathematik 1_, p. 269-271, 1959

4. Kiviat, P.J., Villanueva, R., and Markowitz, H.M., _SIMSCRIPT II.5 Programming Language_, CACI Inc., 1975.

5. Gross, Donald, and Harris, Carl, M., _Fundamentals of Queueing Theory_, p. 52-59, Wiley, 1974.

6. Kleinrock, Leonard, _Queueing Systems Volume 2: Computer Applications_, p. 320-322, Wiley, 1976.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, CA 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93940	2
3.	Department Chairman, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, CA 93940	1
4.	Prof. J. M. Wozencraft, Code 52 Department of Electrical Engineering Naval Postgraduate School Monterey, CA 93940	3
5.	C3 Division (Code D102) Development Center Marine Corps Development Education Command Quantico, VA 22134	2
6.	Prof. Robert G. Gallager Laboratory for Information and Decision Systems Massachusetts Institute of Technology Boston, Ma 01239	1
7.	Dr. Vincent Cerf Information Processing Techniques Office Defense Advanced Research Projects Agency Washington, D.C. 20390	1
8.	Dr. Barry M. Leiner Information Processing Techniques Office Defense Advanced Research Projects Agency Washington, D.C. 20390	1
9.	Naval Electronics Systems Command Code 03 Washington, D.C. 20390	1
10.	Naval Electronics Systems Command Marine Corps Representative, Code PME - 154 Washington, D.C. 20390	1
11.	Office of Naval Research Marine Corps Representative, Code 100M Washington, D.C. 20390	1

12. Commander, Naval Ocean System Center   2
    Code 447 (Library)
    San Diego, CA 92152

13. Capt. Kenneth L. Moore, USAF   1
    Code 62 Mz
    Department of Electrical Engineering
    Naval Postgraduate School
    Monterey, CA 93940

14. PM, Test Measurement and Diagnostics Systems 1
    Attn: DRCPM-TMDS (Maj Robert Heritsch, USA)
    Ft. Monmouth, NJ 07703

15. Lcdr Anthony W. Langerich, USN   3
    Staff, Commander Carrier Group TWO
    FPO New York, NY 09501

16. Lt. George Wasenius, USN   1
    SMC 1162
    Naval Postgraduate School
    Monterey, CA 93940

17. Lt. S. Constandoulakis, HN   1
    Antoniou Paraskeva 5
    Marousi
    Athens, Greece

18. Capt. W.K. Tritschler, USMC   1
    SMC 1162
    Naval Postgraduate School
    Monterey, CA 93940

19. Capt. R. Logan, USMC   1
    SMC 1231
    Naval Postgraduate School
    Monterey, CA 93940

END

FILMED

5 83

DTIC